Spectrum™ Technology Platform

Version 8.0.0 SP3

API Guide

Pitney Bowes Software

© 2013 Pitney Bowes Software Inc. All rights reserved. MapInfo and Group 1 Software are trademarks of Pitney Bowes Software Inc. All other marks and trademarks are property of their respective holders.

USPS® Notices

Pitney Bowes Inc. holds a non-exclusive license to publish and sell ZIP + 4® databases on optical and magnetic media. The following trademarks are owned by the United States Postal Service: CASS, CASS Certified, DPV, eLOT, FASTforward, First-Class Mail, Intelligent Mail, LACS^{Link}, NCOA^{Link}, PAVE, PLANET Code, Postal Service, POSTNET, Post Office, RDI, Suite^{Link} , United States Postal Service, Standard Mail, United States Post Office, USPS, ZIP Code, and ZIP + 4. This list is not exhaustive of the trademarks belonging to the Postal Service.

Pitney Bowes Inc. is a non-exclusive licensee of USPS® for NCOA^{Link}® processing.

Prices for Pitney Bowes Software's products, options, and services are not established, controlled, or approved by USPS® or United States Government. When utilizing RDI™ data to determine parcel-shipping costs, the business decision on which parcel delivery company to use is not made by the USPS® or United States Government.

Centrus Notices

Centrus Data Products contained on this media and used within Centrus applications are protected by various trademarks and by one or more of the following copyrights:

© Copyright United States Postal Service. All rights reserved.

© 2011 TomTom. All rights reserved. TomTom and the TomTom logo are registered trademarks of TomTom N.V.

© Copyright NAVTEQ. All rights reserved

© Copyright United States Census Bureau

© Copyright Nova Marketing Group, Inc.

Portions of this program are © Copyright 1993-2007 by Nova Marketing Group Inc. All Rights Reserved

© Copyright Canada Post Corporation

This CD-ROM contains data from a compilation in which Canada Post Corporation is the copyright owner.

© 2007 Claritas, Inc.

ICU Notices

Copyright © 1995-2011 International Business Machines Corporation and others.

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

GeoNames Notices

The World Geocoding data set contains data licensed from the GeoNames Project (**www.geonames.org**) provided under the Creative Commons Attribution License ("Attribution License") located at **http://creativecommons.org/licenses/by/3.0/legalcode**. Your use of the GeoNames data (described in the Spectrum™ Technology Platform User Manual) is governed by the terms of the Attribution License, and any conflict between your agreement with Pitney Bowes Software, Inc. and the Attribution License will be resolved in favor of the Attribution License solely as it relates to your use of the GeoNames data.

Pitney Bowes Software
Documentation Team
pbbidocs@pb.com

July 16, 2013

Contents

Part I: Introduction....11

Chapter 1: Introduction to Spectrum™ Technology Platform....13

What Is Spectrum™ Technology Platform?....14
Enterprise Data Management Architecture....15
Spectrum™ Technology Platform Architecture....18

Part II: API Fundamentals....23

Chapter 2: Introduction to the API....25

Introduction....26
API Components....26
API Input....28
API Output....28
Sample Applications....29
Network Protocols....29
Client Timeouts....30
General Steps for Using the API....30

Chapter 3: Application Server Support....33

Introduction....34
Java Implementation....34
Client API EJB for WebLogic 8.1.1....43
Client API EJB for WebLogic 9.0.1....46
Client API EJB for WebSphere 5.1.2....51
Client API EJB for WebSphere 6.0.1....54

Chapter 4: The C API.....57
Introduction to the C API.....58
Server.....71
Service.....75
Message.....76
DataTable.....88
DataRow.....96

Chapter 5: The C++ API.....109
Introduction to the C++ API.....110
Server.....121
Service.....124
Message.....125
DataTable.....133
DataRow.....139

Chapter 6: The COM API.....149
Introduction.....150
Server.....153
Service.....156
Message.....157
DataTable.....164
DataRow.....170
Map.....177

Chapter 7: The Java API.....181
Introduction.....182
Server.....185
Service.....191
Message.....192
DataTable.....198
DataRow.....203

Chapter 8: ManagementAPI Methods.....211
Introduction.....212

GetLicenseInfo....212
GetVersionInfo....213

Chapter 9: The .NET API....215
Introduction....216
Server....219
Service....222
Message....223
EnhancedDataTable....229

Chapter 10: Using Web Services....233
Using Web Services....234
The SOAP Interface....235
The REST Interface....237

Part III: Modules....241

Chapter 11: Address Now Module....243
What Is the Address Now Module?....244
BuildGlobalAddress....245
GetGlobalCandidateAddresses....257
ValidateGlobalAddress....264
The ACR Code....289

Chapter 12: Enterprise Geocoding Module....293
What is the Enterprise Geocoding Module?....294
Country Identifier....304
GeocodeAddressAUS....305
GeocodeAddressGBR....332
GeocodeAddressGlobal....350
GeocodeAddressWorld....426
GeocodeUSAddress....464
GNAFPIDLocationSearch....515
ReverseAPNLookup....526
ReverseGeocodeAddressGlobal....539
ReverseGeocodeUSLocation....551

Geocode US Address Auxiliary Files....568
Location Codes for U.S. Geocoding....574
Match Codes for U.S. Geocoding....592
Result Codes for International Geocoding....597
Custom Databases for International Geocoding....602
Encountering False Positives....609

Chapter 13: Enterprise Routing Module....613
What Is the Enterprise Routing Module?....614
GetTravelBoundary....614
GetTravelCostMatrix....624
GetTravelDirections....634
Routing Database Resources....645

Chapter 14: Enterprise Tax Module....647
What Is the Enterprise Tax Module?....648
AssignGeoTAXInfo....651
CalculateDistance....694
Creating a User-Defined Auxiliary File....696

Chapter 15: GeoConfidence Module....701
What Is the GeoConfidence Module?....702
GeoConfidenceSurface....703

Chapter 16: Location Intelligence Module....705
What Is the Location Intelligence Module?....706
Centrus Database Resources....706
Named Connections....707
Named Tables....709
Named Maps....711
Named Tiles....713
User-Defined Centrus Databases....715

Chapter 17: Universal Addressing Module....721
What Is the Universal Addressing Module?....722
AutoCompleteLoqate....734

GetCandidateAddresses....740
GetCandidateAddressesLoqate....750
GetCityStateProvince....756
GetCityStateProvinceLoqate....759
GetPostalCodes....762
GetPostalCodes Loqate....765
ValidateAddress....767
ValidateAddressAUS....834
ValidateAddressGlobal....840
ValidateAddressLoqate....861
Encountering False Positives....886
ValidateAddress Confidence Algorithm....889

Appendices....895

Appendix A: Message XML Schema....897
What Is the Message XML Schema?....898
Message XML Schema Examples....899

Appendix B: Module Matrix....903
Modules, Components, and Databases....904

Appendix C: Country ISO Codes and Module Support....907
Country ISO Codes and Module Support....908

Part I: Introduction

In this section:

- Introduction to Spectrum™ Technology Platform 13

1

Introduction to Spectrum™ Technology Platform

In this section:

- What Is Spectrum™ Technology Platform? 14
- Enterprise Data Management Architecture 15
- Spectrum™ Technology Platform Architecture 18

What Is Spectrum™ Technology Platform?

Spectrum™ Technology Platform is a system that improves the completeness, validity, consistency, timeliness, and accuracy of your data through data standardization, verification and enhancement. Ensuring that your data is accurate, complete, and up to date enables your firm to better understand and connect with your customers.

Note: For more information on Spectrum™ Technology Platform, please visit the **Spectrum™ Technology Platform Video Tutorials** site.

Spectrum™ Technology Platform aids in the design and implementation of business rules for data quality by providing:

- **Parsing, Name Standardization, and Name Validation** on page 14
- **Unique Entity Identification and Consolidation** on page 14
- **Address Validation** on page 14
- **Geocoding** on page 14
- **Location Intelligence** on page 15
- **Tax Jurisdiction Assignment** on page 15

Parsing, Name Standardization, and Name Validation

To perform the most accurate standardization you may need to break up strings of data into multiple fields. Spectrum™ Technology Platform provides advanced parsing features that enable you to parse personal names, company names, and many other terms and abbreviations. In addition, you can create your own list of custom terms to use as the basis of scan/extract operations.

Unique Entity Identification and Consolidation

Identifying unique entities enables you to consolidate records, eliminate duplicates and develop "best-of-breed" records. A "best-of-breed" record is a composite record that is built using data from other records.

Address Validation

Address validation applies rules from the appropriate postal authority to put an address into a standard form and even validate that the address is a deliverable address. Address validation can help you qualify for postal discounts and can improve the deliverability of your mail. For more information, see **What Is the Universal Addressing Module?** on page 722 and **What Is the Address Now Module?** on page 244.

Geocoding

Geocoding is the process of taking an address and determining its geographic coordinates (latitude and longitude). Geocoding can be used for map generation, but that is only one application. The underlying location data can help drive business decisions. Reversing the process, you can enter a geocode (a point represented by a latitude and longitude coordinate) and receive address information about the geocode.

For more information, see **What is the Enterprise Geocoding Module?** on page 294.

Location Intelligence

Location intelligence creates new information about your data by assessing, evaluating, analyzing and modeling geographic relationships. Using location intelligence processing you can verify locations and transform information into valuable business intelligence. For more information, see **What Is the Location Intelligence Module?** on page 706.

Tax Jurisdiction Assignment

Tax jurisdiction assignment takes an address and determines the tax jurisdictions that apply to the address's location. Assigning the most accurate tax jurisdictions can reduce financial risk and regulatory liability.

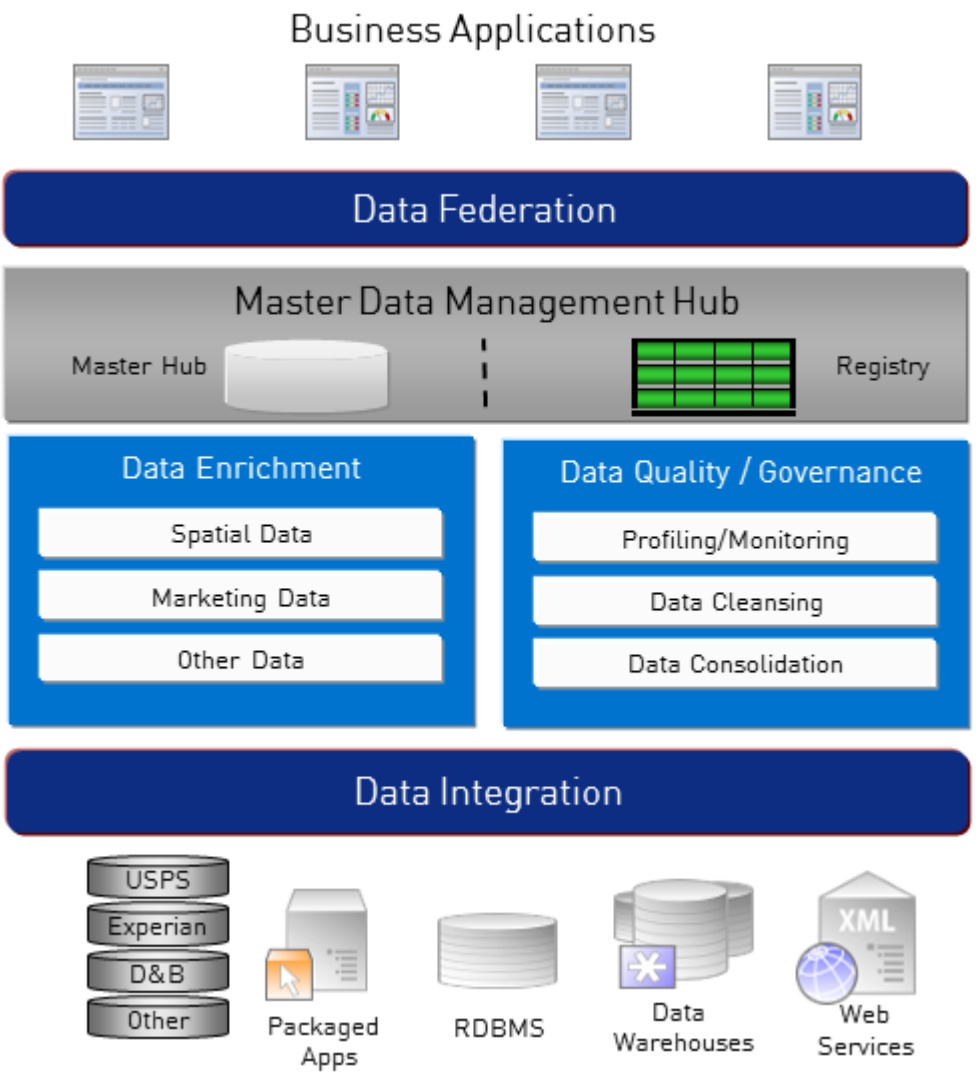
Spectrum™ Technology Platform software from Pitney Bowes Software integrates up-to-date jurisdictional boundaries with the exact street addresses of your customer records, enabling you to append the correct state, county, township, municipal, and special tax district information to your records. Some example uses of tax jurisdiction assignment are:

- Sales and use tax
- Personal property tax
- Insurance premium tax

For more information, see **What Is the Enterprise Tax Module?** on page 648.

Enterprise Data Management Architecture

With Spectrum™ Technology Platform, you can build a comprehensive enterprise data management process, or you can target those individual areas in which your company needs improvement. The following diagram illustrates a complete solution that takes data from its source, through data enrichment and data quality processes, feeding a master data management (MDM) hub which makes a single view of the data available to multiple business applications.



Data Integration

The first step in enterprise data management is to access the data you need to manage. Spectrum™ Technology Platform can connect to data in multiple sources either directly or through integration with your existing data access technologies. It supports batch and real time data integration capabilities for a variety of business needs including data warehousing, data quality, systems integration, and migration. Spectrum™ Technology Platform can access data in RDBMS databases, data warehouses, XML files, flat files, and variable format files. Spectrum™ Technology Platform supports SQL queries with complex joins and aggregations and provides a visual query development tool. In addition, Spectrum™ Technology Platform can access data over REST and SOAP web services.

Spectrum™ Technology Platform can trigger batch processing based on the appearance of one or more source files in a specified folder. This "hot folder" trigger is useful for monitoring FTP uploads and processing them as they occur.

Some of these data integration capabilities require a license for the Enterprise Data Integration Module. For more information, contact your account executive.

Finally, Spectrum™ Technology Platform can integrate with packaged applications such as SAP and Siebel. For more information, see **What is the SAP Module?** and **What is the Siebel Module?**.

Data Quality/Governance

Data quality and data governance processes check your data for duplicate records, inconsistent information, and inaccurate information.

Duplicate matching identifies potential duplicate records or relationships between records, whether the data is name and address in nature or any other type of customer information. Spectrum™ Technology Platform allows you to specify a consistent set of business match rules using boolean matching methods, scoring methods, thresholds, algorithms and weights to determine if a group of records contains duplicates. Spectrum™ Technology Platform supports extensive customization so you can tailor the rules to the unique needs of your business.

Once duplicate records have been identified, you may wish to consolidate records. Spectrum™ Technology Platform allows you to specify how to link or merge duplicate records so you can create the most accurate and complete record from any collection of customer information. For example, a single best-of-breed record can be built from all of the records in a household. The Advanced Matching Module is used to identify duplicates and eliminate them.

Data quality processes also standardize your data. Standardization is a critical process because standardized data elements are necessary to achieve the highest possible results for matching and identifying relationships between records. While several modules perform standardization of one type or another, the Spectrum™ Technology Platform Data Normalization module provides the most comprehensive set of standardization features. In addition, the Universal Name module provides specific data quality features for handling personal name and business name data.

Standardized data is not necessarily accurate data. Spectrum™ Technology Platform can compare your data to known, up-to-date reference data for correctness. The sources used for this process may include regulatory bodies such as the U.S. Postal Service), third-party data providers such as Experian or D&B, or your company's internal reference sources, such as accounting data). Spectrum™ Technology Platform is particularly strong in address data validation. It can validate or standardize addresses in 250 countries and territories around the world. There are two modules that perform address validation:

- Address Now (see **What Is the Address Now Module?** on page 244)
- Universal Addressing (see **What Is the Universal Addressing Module?** on page 722)

To determine which one is right for you, discuss your needs with your account executive.

While Spectrum™ Technology Platform can automatically handle a wide range of data quality issues, there are some situations where a manual review by a data steward is appropriate. To support this, the Business Steward Module provides a way to specify the rules that will trigger a manual review, and it provides a web-based tool for reviewing exception records. It includes integrated access to third-party tools such as Bing maps and Experian data to aid data stewards in the review and resolution process. For more information, see **What Is the Business Steward Module?**.

Data Enrichment

Data enrichment processes augment your data with additional information. Enrichment can be based on spatial data, marketing data, or data from other sources that you wish to use to add additional detail to your data. For example, if you have a database of customer addresses, you could geocode the address to determine the latitude/longitude coordinates of the address and store those coordinates as part of the record. Your customer data could then be used to perform a variety of spatial calculations, such as finding the bank branch nearest the customer. Spectrum™ Technology Platform allows you to enrich your data with a variety of information, including:

- Geocoding (see **What is the Enterprise Geocoding Module?** on page 294)
- Tax jurisdiction assignments (see **What Is the Enterprise Tax Module?** on page 648)
- Geospatial calculations such as finding the nearest site (see **What Is the Location Intelligence Module?** on page 706)
- Driving or walking directions between points (see **What Is the Enterprise Routing Module?** on page 614

Master Data Management Hub

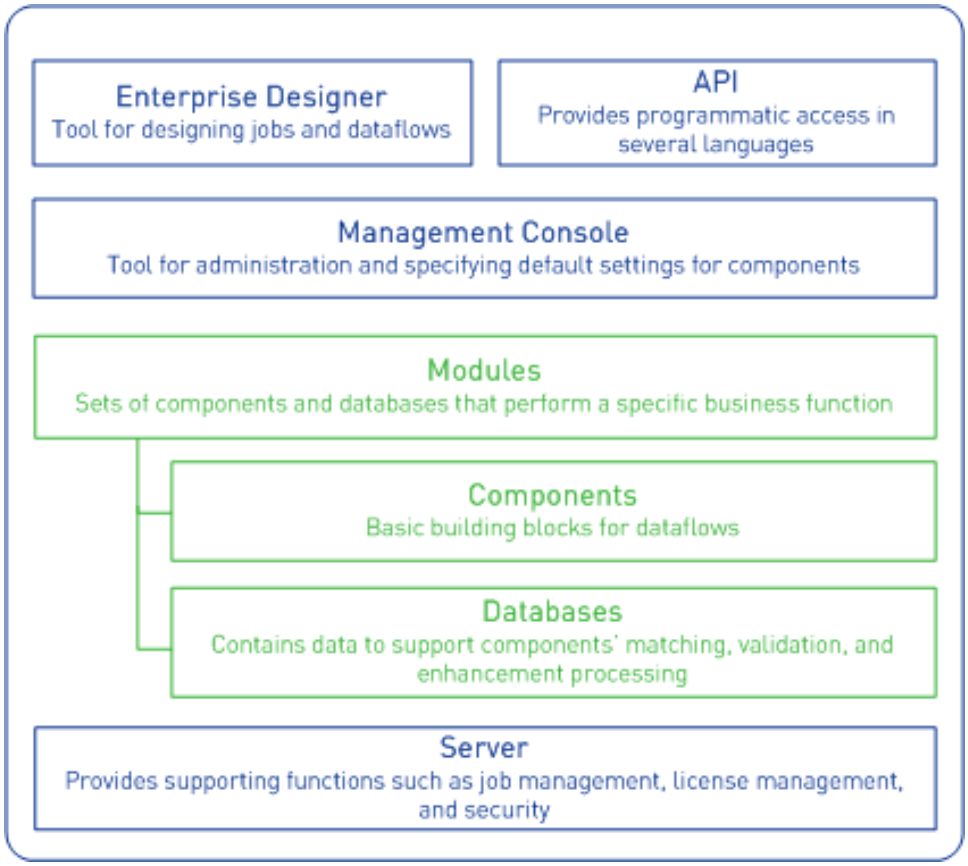
The Master Data Management (MDM) hub allows for rapid modeling of entities and their complex relationships across roles, processes and interactions. It provides built-in social network analysis capabilities to help you understand influencers, predict churn, detect non-obvious relationships and fraudulent patterns, and provide recommendations.

Spectrum™ Technology Platform supports two approaches to the MDM hub. In the master hub approach, the data is maintained in a single MDM database and applications access the data from the MDM database. In the registry approach, the data is maintained in each business application and the MDM hub registry contains keys which are used to find related records. For example, a customer's record may exist in an order entry database and a customer support database. The MDM registry would contain a single key which could be used to access the customer data in both places.

The Data Hub Module provides MDM capabilities. For more information, see **What is the Data Hub Module?**.

Spectrum™ Technology Platform Architecture

Spectrum™ Technology Platform software from Pitney Bowes Software includes a server that supports a number of modules. These modules provide different functions, such as address validation, geocoding, and advanced parsing, among others. The following diagram illustrates the Spectrum™ Technology Platform architecture.


Enterprise Designer
Tool for designing jobs and dataflows
API
Provides programmatic access in several languages
Management Console
Tool for administration and specifying default settings for components
Modules
Sets of components and databases that perform a specific business function
Components
Basic building blocks for dataflows
Databases
Contains data to support components' matching, validation, and enhancement processing
Server
Provides supporting functions such as job management, license management, and security

Server

The foundation of the Spectrum™ Technology Platform is the server. The server handles data processing, synchronizes repository data, and manages communication between the client and the transformation modules via TCP/IP. It provides job management and security features.

Modules

Modules are sets of features that perform a specific function. For example, if you want to standardize your customers' addresses to conform to USPS standards, you would license the Universal Addressing module. If you want to determine the tax jurisdictions that apply to each of your customers, you would license the Enterprise Tax module. You can license just one module or multiple modules, depending on your specific needs. Most modules consist of "components" and databases.

Components

A component is a basic building block in a customer data quality process. Each component performs a specific function. For example, the Enterprise Geocoding module's GeocodeUSAddress component takes an address and returns the latitude and longitude coordinates for that address; the Universal Addressing module's GetCityStateProvince takes a postal code and returns the city and state/province where that postal code is located.

Some components must first be combined with other components into a job, service, or subflow before they can be executed. Use Enterprise Designer to create jobs, services, subflows, and process flows. For more information, see **Enterprise Designer** on page 21.

The components that you have available on your system depend on which Spectrum™ Technology Platform modules you have licensed from Pitney Bowes Software.

Databases

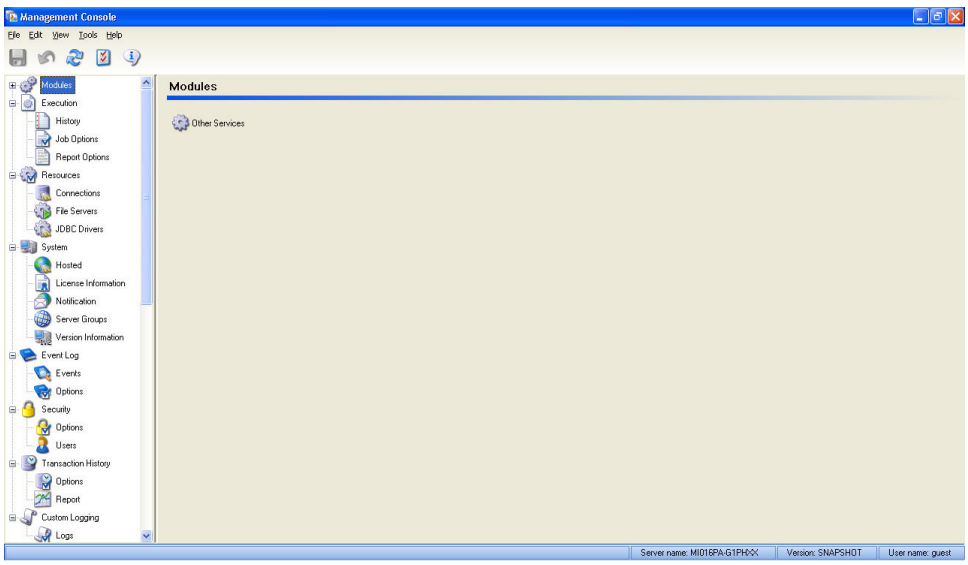
Modules often include databases that contain the data needed by the components in the module. For example, the Universal Addressing module needs to have access to USPS data in order to verify and standardize addresses. So, the Universal Addressing module comes with the U.S. Postal Database, which you must load into a location that is accessible by your Spectrum™ Technology Platform system.

Modules have both required and optional databases. Optional databases provide data needed for certain features that can greatly enhance your Spectrum™ Technology Platform process.

Management Console

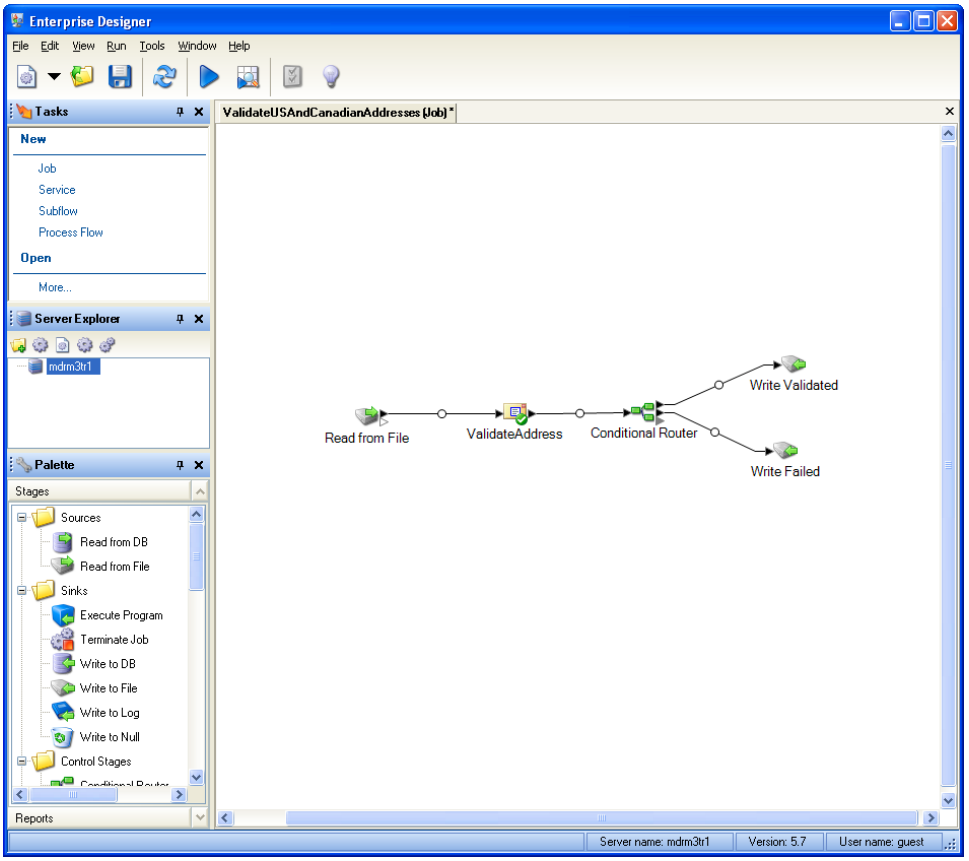
The Management Console is a Windows-based tool for administering Spectrum™ Technology Platform. You can use the Management Console to:

- Specify a server access address
- Select component access method (local or hosted)
- Specify the default settings for Spectrum™ Technology Platform components
- Manage user accounts, including permissions and passwords
- Set up logging, tracking, and reporting.



Enterprise Designer

Enterprise Designer is a Windows-based tool for creating Spectrum™ Technology Platform jobs, services, subflows, and process flows. It utilizes an easy drag-and-drop interface to allow you to graphically create complex dataflows.



API

The Spectrum™ Technology Platform API, which is provided in the Spectrum™ Technology Platform Client API, is designed to provide simple integration, streamline record processing, and support backward compatibility of future versions. The Spectrum™ Technology Platform API can be accessed through:

- C
- C++
- COM
- Java
- .NET
- Web services

For complete information on the Spectrum™ Technology Platform API, see the *Spectrum™ Technology Platform API Guide*.

Part II: API Fundamentals

In this section:

- Introduction to the API 25
- Application Server Support 33
- The C API 57
- The C++ API 109
- The COM API 149
- The Java API 181
- ManagementAPI Methods 211
- The .NET API 215
- Using Web Services 233

2

Introduction to the API

In this section:

- Introduction 26
- API Components 26
- API Input 28
- API Output 28
- Sample Applications 29
- Network Protocols 29
- Client Timeouts 30
- General Steps for Using the API 30

Introduction

The Spectrum™ Technology Platform API is designed to provide simple integration, streamline record processing, and support backward compatibility of future versions. The Spectrum™ Technology Platform API provides the following functions:

- Connecting to the server
- Disconnecting from the server
- Getting the service from the server
- Using the requested service to process a service request message and return a response message.

The following diagram illustrates the input and output data for a service request.

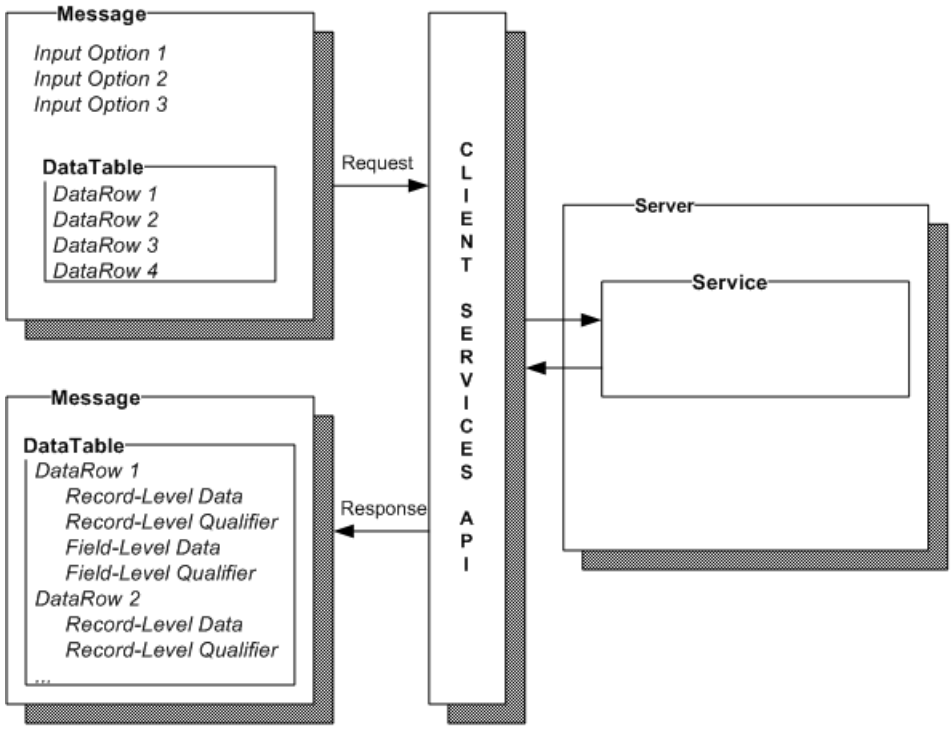


Figure 1: API Input and Output Components

API Components

Despite the inherent differences in the supported programming languages, the API is designed to function in a consistent manner. To do this the API uses these components for all languages:

- Server

- Service
- Message
- DataTable
- DataRow

Server

The Server component represents the Spectrum™ Technology Platform server. Using a Server component, you connect, disconnect, or access a specific service from the server.

- Set connection properties
- Server host
- Server port
- Account ID
- Account password
- Timeout (in milliseconds)
- Message serialization (g1-xml)
- Connect to a server
- Get service
- Disconnect from a server

Service

The Service component is used to process the message you're sending (i.e., send the input message and get back the response). The Service component has just one method: Process message.

Message

Use the Message component to send your input data and receive your output data from the service. You use the get/set methods to get/set various properties. The properties for Message include context properties, such as account ID, account password, service name, and service method; option properties, which are the Service-specific runtime options; and error properties, which are the error class, error message, and error stacktrace.

- GetOption
- SetOption
- GetError
- GetContext
- SetContext
- GetDataTable

DataTable

The DataTable component contains the records for your input and output data. Using the methods associated with this class, you define the column names for your output and add records to the data set. The Reset and Next methods are used to iterate over the results that are returned in a response from the server.

The method names for the DataTable component vary by language. See the details in subsequent chapters for the method names for your API.

DataRow

The DataRow contains schema information and a list of data rows. Individual records reside in rows of data.

The method names for the DataRow component vary by language. See the details in subsequent chapters for the method names for your API.

API Input

There are two types of input: input options and input data.

Input options apply to the records in the DataTable. Options vary from service to service. Spectrum™ Technology Platform allows you to set default processing options through the Management Console. You can also override certain settings for individual requests using the Spectrum™ Technology Platform API. For example, a component like ValidateAddress may be capable of producing output in either mixed case or upper case. In most instances, you need upper case output. However, one of your applications requires output in mixed case. In this situation, you would set the default for casing in the ValidateAddress service to upper case and override the default settings for OutputCasing for that single application. For those requests that should be handled according to the defaults you have specified, it is not necessary to provide any input options in your request. When you have not specified system-level defaults through the Management Console and no processing options are passed at the time of the call, the server uses internal default options. The internal defaults for each service are listed in **Modules** on page 241.

Input data is the data you want to process — an address to validate, for example. Depending on the type of application you are using, you may deal with different types of information. For example, if you are working with address data, the input may be one to four lines of street-level data, plus City, State, Postal Code, Country, and Firm Name.

API Output

For each record, output data contains Record-Level Qualifiers, such as a code reflecting the success or failure of the processing attempt, and record-level data. There are also field-level data. For each field, there may be additional information called Field-Level Qualifiers.

Record-level Output

Data regarding the input data itself (such as the building type or a user-provided key) are called Record-Level Data. Data regarding the processing of the request are referred to as Record-Level Qualifiers. Record-Level Qualifiers include Status of the request (Success, Failure, or Error), Confidence in the accuracy of the output record, and information on failures and errors in processing the record.

Field-level Output

Field-Level Data contains the validated, standardized, or enhanced record. Field-Level Qualifiers include additional data about a given field. For example, the type of a Private Mailbox according to USPS categorization is a field-level qualifier.

Sample Applications

The sample applications provide examples of how to use the API, which is designed to be generic. It uses "Casing" service, which is not distributed with the API. Therefore, you cannot run the sample application unless the casing.car file is in the `<target install directory>/server/app/deploy/` directory. You can also modify the sample application to use one of the services you've licesed, and recompile the sample to run.

Network Protocols

The API is implemented via HTTP, HTTPS, or SOCKET. Spectrum™ Technology Platform typically uses port 8080 to listen for HTTP requests and port 443 for HTTPS requests. Secure Socket Layer (SSL) and Proxy (HTTP, HTTPS) features are also supported in the C, C++, COM, Java, and .NET APIs. .Net, Java, and COM APIs support Unicode; C and C++ APIs have both ASCII and Unicode versions.

In addition to HTTP, Spectrum™ Technology Platform supports a persistent SOCKET connection. The high-speed SOCKET connection provides much faster performance than traditional HTTP. Spectrum™ Technology Platform typically uses port 10119 to listen for SOCKET requests.

To run SSL, the connection type must be set to "HTTPS". The C/C++ corresponding lib directory contains ca-bundle.pem, which contains certificates exported from Microsoft Internet Explorer. This file may be suitable if you are creating a general-purpose application and can't predict which root Certification Authority (CA) will be used by the servers your application will connect to. If you know the specific root CA, then it will be more secure to specify just that one certificate in your CA bundle file. You need to copy "ca-bundle.pem" from the following locations to your working directory:

- `/platforms/unix/xxx/32/lib/ (where "xxx" represents the flavor of Unix you are running)`
- `/platforms/windows/com/lib/`
- `/platforms/windows/cpp/32/lib/v6/g1`

To run ASP with HTTPS, copy the `ClientAPI/platforms/windows/com/lib/ca-bundle.pem` file to your working directory. For example:

- If you use Internet Information Services to run ASP, copy ca-bundle.pem to the Windows system directory (e.g., `C:\WINNT\system32`).
- If you use Internet Explorer to run ASP, copy ca-bundle.pem to the Internet Explorer default working directory (e.g., `C:/Documents and Settings/<user>/Desktop`).

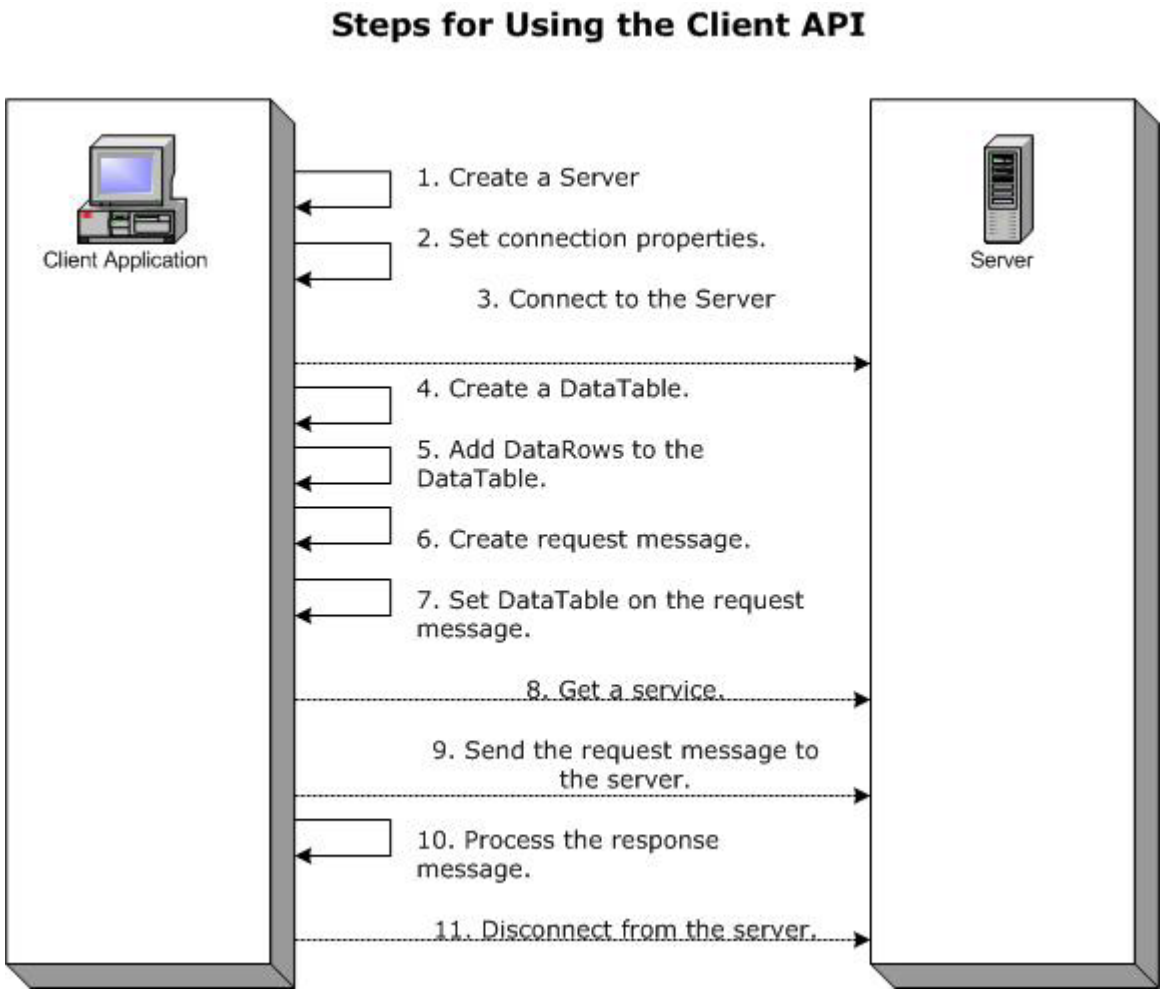
Client Timeouts

If you experience timeouts between the client and server, you can increase the timeout value for the client. Use the setConnectionProperty ("timeout", "your value") method to set this value. In the sample application, locate the connection.properties file in the <target install directory>\common\data directory. Find the default timeout value (called "timeout") and adjust it accordingly. The value is in milliseconds.

General Steps for Using the API

The basic steps for using the Spectrum™ Technology Platform API are listed below and are illustrated in Figure 1-2.

1. Create a Server instance.
2. Set connection properties (connection type, host, port, etc.).
3. Connect to the Server.
4. Create a DataTable.
5. Add records to the DataTable.
6. Create a request message.
7. Set DataTable on the request message.
8. Get a service.
9. Send the request message to the Server.
10. Process the response message.
11. Disconnect from the Server.


Steps for Using the Client API
Client Application
Server
1. Create a Server
2. Set connection properties.
3. Connect to the Server
4. Create a DataTable.
5. Add DataRows to the DataTable.
6. Create request message.
7. Set DataTable on the request message.
8. Get a service.
9. Send the request message to the server.
10. Process the response message.
11. Disconnect from the server.

3

Application Server Support

In this section:

- Introduction 34
- Java Implementation 34
- Client API EJB for WebLogic 8.1.1 43
- Client API EJB for WebLogic 9.0.1 46
- Client API EJB for WebSphere 5.1.2 51
- Client API EJB for WebSphere 6.0.1 54

Introduction

You can invoke the Spectrum™ Technology Platform Client API using a generic EJB implementation according to Sun J2EE EJB specifications. The following table shows locations for items related to application server support:

Table 1: Application Server Support Locations

/ejb/common/src/	Contains source code for business logic implementation and application server independent EJB implementation.
/ejb/common/doc/	Contains documentation for application server independent EJB implementation.
/ejb/common/doc/api/	Contains javadoc APIs for all classes.
ejb/common/doc/common.pdf	Contains a PDF file that discusses building, deploying, and using EJB; also includes code snippets.
ejb/common/conf	Contains deployment descriptors for application server independent EJB implementation.
ejb/common/build	Contains an Ant build script for application server independent EJB.

Java Implementation

The Java implementation of the Client API contains the following methods in com.g1.dcg.client.Server:

Table 2: Java Methods

public void setConnectionProperty(String name, String value)	Allows you to set connection property on Server object.
public void connect()	Connects to Spectrum™ Technology Platform.
public Service getService(String serviceName)	Returns the Service object, providing access to services

In turn, com.g1.dcg.client.Service provides the following interface:

public Message process(Message request)	Processes the request and returns results.

Client API Exposed via Session Bean

Spectrum™ Technology Platform contains an API in the form of a reusable deployable component that can be deployed in a J2EE-compliant EJB container and that provides an API similar to the one exposed by the Spectrum™ Technology Platform Client API. The reusable deployment component is implemented in a form of Session Bean—in particular, Stateless Session Bean (and later in the form of Message-Driven Bean). The Session Bean exposes a collection of related methods used to access back-end services, data, and APIs inside EJB container.

Internally, a Session Bean is composed of the following parts:

Table 3: Session Bean Parts

The Component Interface	The client view of the bean. It contains all the "business methods" of the bean. The Spectrum™ Technology Platform Component Interface contains only one method— process(...)—that exposes the service.
The Home Interface	Contains all the methods for the bean lifecycle used by the client application—for example, Home Interface provides the method that allows you to create a new instance of the bean.
The bean implementation class	Implements business methods and all methods mandated by the J2EE EJB specification, allowing the bean to be managed by the EJB container.
The deployment descriptor	Contains bean properties used by the EJB container and by the bean implementation class.

According to the J2EE EJB 2.0 specification, the couple "Component Interface and Home Interface" may be either local or remote. Local interfaces (Home Interface and Component Interface) are to be used by a client application running inside the same JVM as the EJB component. Remote Interfaces (also Home Interface and Component Interface) are to be used by a client application running outside EJB component's JVM. Create and finder methods of local or remote home interfaces return local or remote component interfaces respectively. An EJB component can have one or both of those interfaces at the same time; the Spectrum™ Technology Platform EJB provides both local and remote interfaces.

A session bean object is a short-lived object that executes on behalf of a single client. There are stateful and stateless session beans; the stateful session bean maintains the state across method calls, while

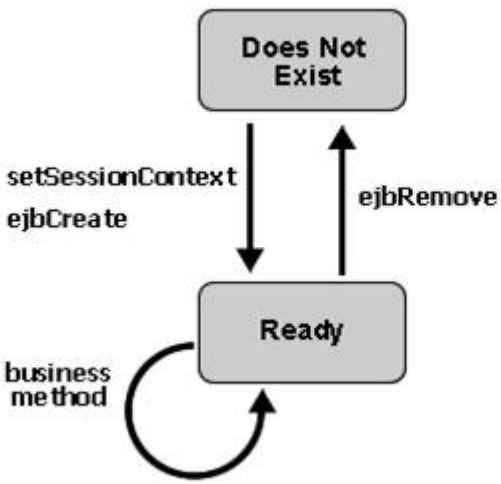
the stateless session bean does not maintain that state. The Spectrum™ Technology Platform interface is presented in form of a Stateless Session Bean.

Life Cycle of the Stateless Session Bean

A bean goes through various stages during its life cycle. Each type of enterprise bean (stateful session, stateless session, entity, message-driven) has a different life cycle.

The stateless session bean life cycle starts with the EJB container instantiating the bean implementation class and then invoking setSessionContext and ejbCreate methods in the session bean, passing session context into setSessionContext. The bean is then ready to have its business methods invoked.

The diagram below depicts the life cycle of the stateless session bean.



At the end of the life cycle, the EJB container invokes ejbRemove on bean instance and then the stateless session bean instance is ready for the garbage collection.

Note: The client application has no control over the stateless session bean life cycle, except during the invocation of the business methods of the bean.

The Home Interface

A session bean's home interface defines one or more create(...) methods. Each create(...) method must be named "create" and its signature must match one of ejbCreate methods defined in the EJB bean implementation class. For example, if the bean implementation class has a method ejbCreate(String id), then the home interface must define a method create(String id). The return type of the create(...) method must be the remote interface type for this Session Bean. In addition to custom create(...) methods, the home interface must have one create method that has no arguments.

All exceptions defined in the throws clause of an ejbCreate(...) method in the bean implementation class must be defined in the throws clause of the matching create method of the home interface.

A remote home interface extends the javax.ejb.EJBHome interface, while a local home interface extends the javax.ejb.EJBLocalHome interface.

The remote version of Home Interface for Client API Bean contains the following methods:

```
public interface ClientApiHome extends javax.ejb.EJBHome
{
 public ClientApi create() throws CreateException, RemoteException;
}
The local version of Home Interface for Client API Bean is defined as follows:

public interface LocalClientApiHome extends javax.ejb.EJBLocalHome
{
 public ClientApi create() throws CreateException;
}
```

The Component Interface

The Component Interface is the client's view of an instance of a session bean. This interface contains business methods of the bean; it must extend the javax.ejb.EJBObject interface if it is remote, and javax.ejb.EJBLocalObject if it is local. The methods defined in a remote component interface must follow the rules for Java RMI (meaning that their arguments and return value must be valid types for Java RMI, and their throws clause must include java.rmi.RemoteException). For each method defined in the Component Interface, there must be a matching method in the session bean implementation class. A matching method is a method that has same name, same arguments list, same return type, and same exceptions list. One exception is the public interface ClientApi, which extends javax.ejb.EJBObject.

The remote version of the Component Interface for Client API Bean contains the following methods:

```
{
 public Message process(String service, Message message) throws
MessageProcessingException;
}
```

The local version of Component Interface for Client API Bean is defined as follows:

public interface LocalClientApi extends javax.ejb.EJBLocalObject

```
{
 public Message process(String service, Message message) throws
MessageProcessingException;
}
```

As is evident in these methods, both versions of Component Interface include a process(...) method. This method will be used by a client application to send data to a service and receive processed results back.

The Enterprise Bean Implementation Class

The bean implementation class implements the business methods of the Component Interface and the methods of the SessionBean interface required by J2EE EJB container. The bean implementation class must be defined as public and cannot be abstract. The SessionBean methods that the bean implementation class must implement are the following:

Table 4: Session Bean Methods

public void setSessionContext(SessionContext ic)	This method is used by the container to pass a reference to the SessionContext to the bean instance. The container invokes this method after the instance has been created. Generally, this method stores a reference to the session context in an instance variable.
public void ejbRemove()	This method is invoked by the container when the instance is in the process of being removed by the container. Since most session beans do not have any resource state to clean up, the implementation of this method is typically left empty for the Session Bean.
public void ejbPassivate()	This method is invoked by the container when the container wants to passivate the instance. After this method completes, the instance must be in a state that allows the container to use the Java serialization protocol to externalize and store the instance's state.
public void ejbActivate()	This method is invoked by the container when the instance has been reactivated. The instance should acquire any resource that it has released during the passivation stage.

In addition to these methods, the bean implementation class must implement methods from Component Interface; for Spectrum™ Technology Platform, the case Client API Bean must provide implementation for the process(...) method.

The following shows the implementation of the Client API Bean:

```
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.EJBObject;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.SessionSynchronization;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class ClientApiBean implements SessionBean {
 protected SessionContext ctx = null;
 private Server server = null;
 public void ejbCreate()
 {
  // load configuration properties here and connect to the server
  loadConfigurationProperties(server);
  server.connect();
 }
```

```
 public void ejbActivate() {
  // we leave implementation of this method empty
 }
 public void ejbPassivate() {
  // we leave implementation of this method empty
 }
 public void ejbRemove() {
  // close connection to the server
 }
 public void setSessionContext(SessionContext ctx) {
  this.ctx = ctx;
 }
 public Message process(String service, Message message) {
  Service service = server.getService(service);
  return service.process(message);
 }
}
```

The following shows the implementation of each non-trivial method in the Client API Bean:

Table 5: Client API Bean Implementation

public void ejbCreate()	The ejbCreate method is called when a new instance of ClientApiBean is created inside an EJB container. It is called exactly once in a life cycle of a Stateless Session Bean, so we use it to load configuration properties (server name, port, user name, and password), configure the server connection, and connect to the server.
public void ejbRemove()	The ejbRemove method invoked by EJB container right before the container discards that bean instance. We use that method to close and cleanup connection to the server.
public Message process(Message message) throws MessageProcessingException	This is the actual business method of the bean; it specifies the service and then calls the process method of Spectrum™ Technology Platform Client API.

Client API Exposed via Message-Driven Bean

In addition to using Stateless Session Bean to expose Spectrum™ Technology Platform Client API in a J2EE environment, Spectrum™ Technology Platform provides implementation in the form of Message-Driven Bean. Message-Driven Bean provides server-side business functionality, namely functionality of the Client API, in response to an asynchronous message inside the EJB container.

The communication path from a client to an MDB is:

1. The client sends a message to a Java Message Service (JMS) destination, which is associated with an EJB container.

2. When a message arrives in a queue associated with an MDB, the container passes the message to the onMessage() method on an MDB instance.
3. The MDB instance then performs the requested service using the message as its input.

A client can be any JMS message sender—not just an application client or servlet. The message destination can be either a JMS Queue or a JMS Topic. JMS ensures the delivery of the message to the destination and automatically handles acknowledgements, retries, and transactions.

The primary differences between Stateless Session Beans and MDBs include the following:

- Service requests to a stateless session bean are synchronous (by calling methods on a component interface). Service requests to an MDB are asynchronous (by sending a JMS message to message destination).
- A stateless session bean can return data directly to a caller as the return value of a component interface method invocation. While the JMS destination of an MDB can acknowledge receipt of a message, the component can not return data to the sender of the message. If an MDB wants to send information back to the sender, it must send another JMS message as a reply.
- Only valid EJB clients (such as J2EE application clients and Web components such as servlets and JSP pages) can access stateless session beans. By contrast, any JMS sender can invoke an MDB by sending a message to that bean's JMS destination.
- A session bean has a client security context that controls which clients can create and use the bean. MDBs have no client security context. Access to an MDB is managed by controlling access to the MBD's JMS destination.

Internally, Message-Driven Bean is composed of a single class—a Message-Driven Bean class. The MDB class must meet the following requirements:

- It must be public and must not be abstract or final. It must not implement a finalize method.
- It must have a public, zero-argument constructor. J2EE services (such as access to JNDI) are not available in that constructor. MDB instances should be initialized in the ejbCreate method instead of this constructor.
- It must have a single public void method, ejbCreate(), which takes no arguments. ejbCreate() is called after an MDB instance has been created, and also after setMessageDrivenContext is called. The purpose of ejbCreate is to allow the bean to initialize its internal state. In Spectrum™ Technology Platform, ejbCreate will load the server connection properties and configure Server instance.
- It must implement the interface javax.ejb.MessageDrivenBean, which has two methods:
- setMessageDrivenContext: The EJB container calls setMessageDrivenContext when it creates a new MDB instance, passing an argument of type javax.ejb.MessageDrivenContext. The MessageDrivenContext interface is analogous to the EntityContext and SessionContext interfaces of entity and session beans. The MDB usually retains a copy of the object and uses it later to request services from the container. Because MDBs do not have home interfaces, the MessageDrivenContext methods getEJBHome and getEJBLocalHome are meaningless and should not be called.
- ejbRemove: The EJB container calls ejbRemove before removing an MDB instance. This method allows the bean to clean up any resources it might hold, such as closing the connection to Spectrum™ Technology Platform. The container may automatically remove MDB instances as part of its resource management mechanism.
- It must implement javax.jms.MessageListener. The interface has a single method:
- onMessage: An MDB's onMessage method implements application-specific business functionality—sending messages to the server for processing and sending results back to the client. This is the method that receives Message objects from a JMS destination.

All application-specific functionality in the MDB goes into the onMessage() method. The other methods provide support for the bean's life cycle events.

The following shows the implementation of the Client API Message-Driven Bean:

```
public class ClientApiMDB implements MessageDrivenBean, MessageListener
{
...
 public void onMessage(javax.jms.Message message)
 {
  try
  {
   ObjectMessage msg = (ObjectMessage)message;
   String service = msg.getStringProperty(SERVICE_PROPERTY);
   com.g1.dcg.message.Message request =
(com.g1.dcg.message.Message)msg.getObject();
   Service serviceObj = server.getService(service);
   com.g1.dcg.message.Message response = serviceObj.process(request);
   processResponse(response);
  }
  catch(JMSException jmse)
  {
    // TODO: log the exception using your app server logging mechanism
  }
  catch(NamingException ne)
  {
    // TODO: log the exception using your app server logging mechanism
  }
  catch(ServerException se)
  {
    // TODO: log the exception using your app server logging mechanism
  }
  catch(MessageProcessingException mpe)
  {
    // TODO: log the exception using your app server logging mechanism
  }
 }
...
}
```

The message-driven bean contains the onMessage() method that implements all business logic. First, the message passed into the onMessage method gets cast down to ObjectMessage type. To determine the name of the service to be used, we retrieve the string property from the message. The object contained in the ObjectMessage is our Message object; we get the Service instance from the Server object using the service name and then process the request and put it back into the response queue.

Deployment Descriptors

Deployment descriptors contain metadata about enterprise beans contained in a deployment unit. Deployment descriptors for stateless session bean and message-driven bean are both contained in file ejb-jar.xml. Here is the structure of the file:

```
<ejb-jar>
<display-name>J2EE Client API</display-name>
<enterprise-beans>
 <session>
  <description>Stateless Session Bean to access Client API</description>
  <ejb-name>ClientApi</ejb-name>
  <home>com.g1.dcg.client.ejb.stateless.ClientApiHome</home>
  <remote>com.g1.dcg.client.ejb.stateless.ClientApi</remote>
  <ejb-class>com.g1.dcg.client.ejb.stateless.ClientApiBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
```

```
 </session>
 <message-driven>
  <description>Message-Driven Bean to access Client API</description>
  <display-name>ClientApiMDB</display-name>
  <ejb-name>ClientApiMDB</ejb-name>
  <ejb-class>com.g1.dcg.client.ejb.mdb.ClientApiMDB</ejb-class>
  <transaction-type>Container</transaction-type>
  <message-driven-destination>
   <destination-type>javax.jms.Queue</destination-type>
  </message-driven-destination>
 </message-driven>
</enterprise-beans>
</ejb-jar>
```

The <session> element contains the deployment descriptor for the stateless session bean. Deployment descriptor contains a description and display name of the bean, the ejb-name, and all classes and interfaces comprising the bean. The session-type element specifies the type of the session bean—in this case, stateless. The transaction-type element specifies container-managed transactions.

The next element inside <enterprise-beans> is a <message-driven> element. It describes the deployment of a message-driven bean and includes a description and display name of the bean, the ejb-name to be used for locating the bean, and the implementation class for the bean. The message-driven destination specifies the destination type the message-driven bean receives messages from.

Deployment descriptor also contains a mechanism for specifying EJB environment variables, which are named variables stored in the deployment descriptor that can be accessed by the enterprise java bean at runtime. EJB environment variables are scoped to the enclosing enterprise bean definition, so different beans may have the same variable defined within their scopes.

EJB environment variables are defined in the <env-entry> element inside corresponding bean definition:

```
<env-entry>
  <env-entry-name>connection.type</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>http</env-entry-value>
</env-entry>
```

This entry defines a variable with the name "connection.type" of type String that has value "http". Client API Bean uses EJB environment variables to specify connection information for connecting to remote Spectrum™ Technology Platform.

Building Spectrum™ Technology Platform Java Beans

After you develop the beans they are ready to be compiled and packaged into deployable units.

To build EJB components we use Ant, which can be obtained from http://ant.apache.org. The build script used by Ant, build.xml, contains the following targets:

Table 6: EJB Targets

clean	used to clean the class files and jar/war files created during the build
prepare	used to create the directory structure for the build
compile	used to compile Java classes

jar	used to create jar files containing Client API EJBs

For example, to build a jar file containing Client API EJBs, issue the following command:

$ ant jar

Ant will create build directories, compile Java classes, and create JAR file containing necessary configuration and binary files.

Note: Be sure to specify the connection parameters for both stateless session beans and message-driven beans in ejb-jar.xml, according to your server configuration.

Deploying EJB Using the Sun Reference Implementation

Before proceeding, you need to download and install Sun Reference Implementation of EJB Container at **www.oracle.com/technetwork/java**.

Follow the instructions in the current version of Sun Reference Implementation for deploying clientapi.jar into the application server.

Building a Client Application for the EJB Client API

To test the Client API deployed in the EJB Container, use a simple web application displaying a web page that allows you to specify the service name, field name, and field value; pick one of the three interfaces to use (stateless session bean with local interface, stateless session bean with remote interface, or message-driven bean); and send a request to the servlet. In turn, the servlet will process the request and call the business method on a Client API Bean (or send a JMS message in case of message-driven bean), and display the content of response in HTML format.

Deploying the Client Application for the EJB Client API

To build and deploy the client application, use the target "war" in the build.xml Ant build script. This will produce a clientapi.war file, which can then be deployed into Sun Reference Implementation.

Please follow the instructions in the current version of Sun Reference Implementation for deploying clientapi.war into the application server.

Client API EJB for WebLogic 8.1.1

Packaging Spectrum™ Technology Platform Client API EJB

Before you can use the Client API EJB within the WebLogic application server, it needs to be packaged and deployed in accordance with WebLogic requirements. There are compiled class files for stateless session bean and message-driven bean implementations of Client API EJB, as well as Java classes for servlet. Now we need to create BEA WebLogic-specific deployment descriptors, create a package application, and deploy it to WebLogic Server.

In addition to the standard deployment descriptor contained in ejb-jar.xml, the WebLogic Server also requires a WebLogic-specific deployment descriptor. The deployment descriptor enables us to configure parameters that are specific to WebLogic implementation. The WebLogic-specific deployment descriptor is contained in weblogic-ejb-jar.xml, which is located in the same directory with ejb-jar.xml.

Following is an XML fragment used to describe stateless session bean implementation of Client API EJB:

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>ClientApi</ejb-name>
    <jndi-name>ejb/RemoteClientView/ClientApi</jndi-name>
  <weblogic-enterprise-bean>
  <weblogic-enterprise-bean>
    <ejb-name>ClientApi</ejb-name>
    <local-jndi-name>ejb/LocalClientView/ClientApi</local-jndi- name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

This simple WebLogic deployment descriptor contains only two pieces of information for both local and remote implementation. First, it uses an ejb-name tag to specify that these parameters are for the Client API EJB. This enables the container to associate these values with the parameters read from the ejb-jar.xml descriptor. The second tag is the Java Naming and Directory Interface (JNDI) name. When the stateless session bean implementation of Client API is deployed in the WebLogic Server, the EJB container binds the home interface into the JNDI tree using the <jndi-name> (or <local-jndi-name> for local component interface) specified in the weblogic-ejb-jar.xml. Clients can then find the EJB by using JNDI to look up the name "ejb/RemoteClientView/ClientApi".

A message-driven bean will require its own section in weblogic-ejb-jar.xml descriptor file, having the root element <weblogic-enterprise-bean>, which is the same as a stateless session bean.

The only required element for the message-driven bean is the <destination-jndi-name>. When the EJB container deploys the message-driven bean, it uses JNDI name to look up resource references such as the JMS destinations. Unlike stateless session bean, a <jndi- name> for the message-driven bean is not required since there is no direct client interaction.

Following is an XML fragment describing the message-driven bean implementation of Client API:

```
<weblogic-enterprise-bean>
  <ejb-name>ClientApiMDB</ejb-name>
  <message-driven-descriptor>
    <destination-jndi-name>
      jms/G1QueueDestinationName
    </destination-jndi-name>
  </message-driven-descriptor>
</weblogic-enterprise-bean>
```

Spectrum™ Technology Platform includes both Enterprise Java Beans and a Web application. We use a single enterprise archive (EAR) file to package these components into a single distribution file, which then can be deployed in WebLogic Server.

Enterprise archive contains an EJB JAR file, a Web Application WAR file, Client API libraries (g1client.jar and g1message.jar), and deployment descriptors specific to enterprise archive. Enterprise archive deployment descriptors include application.xml (app server- independent deployment descriptor) and weblogic-application.xml (WebLogic-specific application deployment descriptor).

Application server independent deployment descriptor contains a display-name and description for the application, a description of each module, and configuration parameters for the web application. This XML fragment provides the content of application.xml file:

```
<application>
  <display-name>ClientAPI</display-name>
  <description>Spectrum™ Technology Platform Client API EJB </description>
  <module>
    <ejb>clientapi.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>clientapi.war</web-uri>
      <context-root>/clientapi</context-root>
    </web>
  </module>
</application>
```

The previous configuration describes the application ClientAPI, which contains two modules: EJB module contained in clientapi.jar and Web application contained in clientapi.war, which is going to be mapped to context root /clientapi.

WebLogic-specific application deployment descriptor provides a number of configuration options, including caching configuration, parsers, classloader configuration, application parameters, and others. Please refer to your WebLogic 8.1 Server documentation for more information.

Spectrum™ Technology Platform contains an empty weblogic-application.xml that contains the following XML fragment:

```
<weblogic-application>
</weblogic-application>
```

WebLogic provides a location within an EAR file where you can store shared utility classes, such as classes contained in the Client API. The JAR files for those classes should be placed in the APP-INF/lib directory and individual classes (if any) should be placed into APP-INF/classes directory. These classes are loaded into the root classloader for the application.

This feature is specific to WebLogic and allows you to package utility libraries within the EAR instead of adding to the system classpath. Another option, including referencing the JAR files in manifest inside EJB JAR, does not work since WebLogic 8.1 does not honor the Class-Path declaration within the manifest inside EJB JAR.

Spectrum™ Technology Platform Client API requires two JAR files to be present in the application: g1client.jar and g1message.jar. Both of them are stored in APP-INF/lib directory in clientapi.ear archive.

Deploying Client API EJB Application in WebLogic Server

Before deployment, ensure that the associated JMS destination exists. For our message-driven bean implementation configuration, the server should have a JMS queue deployed with JNDI name MessageQueue. If the JMS destination does not exist, it needs to be created.

Follow the steps below to create a JMS destination associated with JMS queue in WebLogic Server:

1. Create a directory to use as a JMS file store in the file system.
2. Create a directory called g1_store at the root level of your WebLogic installation.

3. Configure the JMS file store using the WebLogic Server administration console. Select Services > Stores on the console's home page.
4. Configure a JMS server using the WebLogic administration console. All JMS-administered objects such as destinations or connection factories exist within a JMS Server. Each WebLogic Server instance can include multiple JMS Server instances.
5. Deploy the JMS server on a WebLogic Server instance.
6. Select JMS > Destinations. The application looks up the destination using JNDI; we create destination jms/G1QueueDestinationName.
7. Click Configure DestinationsÂ… at the bottom of JMS Server configuration page.

   After building the EAR file and configuring the JMS destination to be used by Client API message-driven bean, you need to deploy the EAR on WebLogic Server.
8. Go to the WebLogic administrative console and select Domain Configuration > Applications > Your Deployed Resources.
9. Click Deploy New Application.

   Note: Before deploying an application it should be uploaded to WebLogic Server. Select upload your file(s) link to upload the application. When the upload dialog box appears, go to the location of clientapi.ear and upload the file.
10. Select clientapi.ear and click Continue. You will see a screen that shows the deployment parameters.

    After successfully deploying an application, you should see deployment status for EJB modules and for Web application modules; if there were any errors during deployment they would be displayed on the same screen.

    The next step you should take is to run the client web application for the Spectrum™ Technology Platform Client API (EJB).
11. Open the browser and enter the following URL: http://localhost:7001/clientapi/. If you are running the WebLogic Server on localhost and it listens to HTTP requests on port 7001, you should see a configuration page.
12. Enter the following information:

    - service name
    - column name for the request
    - value for that column
13. Click Submit. You will be taken to the page that displays results of processing the service.

Client API EJB for WebLogic 9.0.1

Packaging Spectrum™ Technology Platform Client API EJB

Before you can use the Client API EJB within the WebLogic application server, it needs to be packaged and deployed in accordance with WebLogic requirements. There are compiled class files for stateless session bean and message-driven bean implementations of Client API EJB, as well as Java classes for Web application. Now we need to create BEA WebLogic-specific deployment descriptors, create a package application, and deploy it to WebLogic Server.

Building WebLogic-Specific Deployment Descriptors for Client API EJBs

In addition to the standard deployment descriptor contained in ejb-jar.xml, the WebLogic Server also requires a WebLogic-specific deployment descriptor. The deployment descriptor enables us to configure parameters that are specific to WebLogic implementation. The WebLogic-specific deployment descriptor is contained in weblogic-ejb-jar.xml, which is located in the same directory with ejb-jar.xml.

Following is an XML fragment used to describe stateless session bean implementation of Client API EJB:

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>ClientApi</ejb-name>
    <jndi-name>ejb/RemoteClientView/ClientApi</jndi-name>
  <weblogic-enterprise-bean>
  <weblogic-enterprise-bean>
    <ejb-name>ClientApi</ejb-name>
    <local-jndi-name>ejb/LocalClientView/ClientApi</local-jndi- name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

This simple WebLogic deployment descriptor contains only two pieces of information for both local and remote implementation. First, it uses an ejb-name tag to specify that these parameters are for the Client API EJB. This enables the container to associate these values with the parameters read from the ejb-jar.xml descriptor. The second tag is the Java Naming and Directory Interface (JNDI) name. When the stateless session bean implementation of Client API is deployed in the WebLogic Server, the EJB container binds the home interface into the JNDI tree using the <jndi-name> (or <local-jndi-name> for local component interface) specified in the weblogic-ejb-jar.xml. Clients can then find the EJB by using JNDI to look up the name "ejb/RemoteClientView/ClientApi" or ejb/LocalClientView/ ClientApi" for local component interface, respectively.

Message-driven bean will require its own section in weblogic-ejb-jar.xml descriptor file, having the root element <weblogic-enterprise-bean>, which is the same as stateless session bean.

The only required element for message-driven bean is the <destination-jndi-name>. When the EJB container deploys the message-driven bean, it uses JNDI name to look up resource references such as the JMS destinations. Unlike stateless session bean, a <jndi-name> for the message-driven bean is not required since there is no direct client interaction.

Following is an XML fragment describing the message-driven bean implementation of Client API:

```
<weblogic-enterprise-bean>
  <ejb-name>ClientApiMDB</ejb-name>
  <message-driven-descriptor>
    <destination-jndi-name>
      jms/G1QueueDestinationName
    </destination-jndi-name>
  </message-driven-descriptor>
</weblogic-enterprise-bean>
```

Packaging Application Archive (EAR) File

Spectrum™ Technology Platform includes both Enterprise Java Beans and a Web application. We use a single enterprise archive (EAR) file to package these components into a single distribution file, which then can be deployed in WebLogic Server.

Enterprise archive contains an EJB JAR file, a Web Application WAR file, Client API libraries (g1client.jar and g1message.jar), and deployment descriptors specific to enterprise archive. Enterprise archive deployment descriptors include application.xml (app server- independent deployment descriptor) and weblogic-application.xml (WebLogic-specific application deployment descriptor).

Building Generic and WebLogic-Specific Application Deployment Descriptors

Application server independent deployment descriptor contains a display-name and description for the application, a description of each module, and configuration parameters for the web application. This XML fragment provides the content of application.xml file:

```
<application>
  <display-name>ClientAPI</display-name>
  <description>Spectrum™ Technology Platform Client API EJB </description>
  <module>
    <ejb>clientapi.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>clientapi.war</web-uri>
      <context-root>/clientapi</context-root>
    </web>
  </module>
</application>
```

The previous configuration describes the application ClientAPI, which contains two modules: EJB module contained in clientapi.jar and Web application contained in clientapi.war, which is going to be mapped to context root /clientapi.

WebLogic-specific application deployment descriptor provides a number of configuration options, including caching configuration, parsers, classloader configuration, application parameters, and others. Please refer to your WebLogic 9.0 Server documentation for more information.

Spectrum™ Technology Platform contains an empty weblogic-application.xml that contains the following XML fragment:

```
<weblogic-application>
</weblogic-application>
```

Adding Spectrum™ Technology Platform Client API Classes and Libraries

WebLogic provides a location within an EAR file where you can store shared utility classes, such as classes contained in the Spectrum™ Technology Platform Client API. The JAR files for those classes should be placed in the APP-INF/lib directory and individual classes (if any) should be placed into APP-INF/classes directory. These classes are loaded into the root classloader for the application.

This feature is specific to WebLogic and allows you to package utility libraries within the EAR instead of adding the to system classpath. Another option, including referencing the JAR files in manifest inside EJB JAR, does not work since WebLogic 9.0 does not honor the Class-Path declaration within the manifest inside EJB JAR.

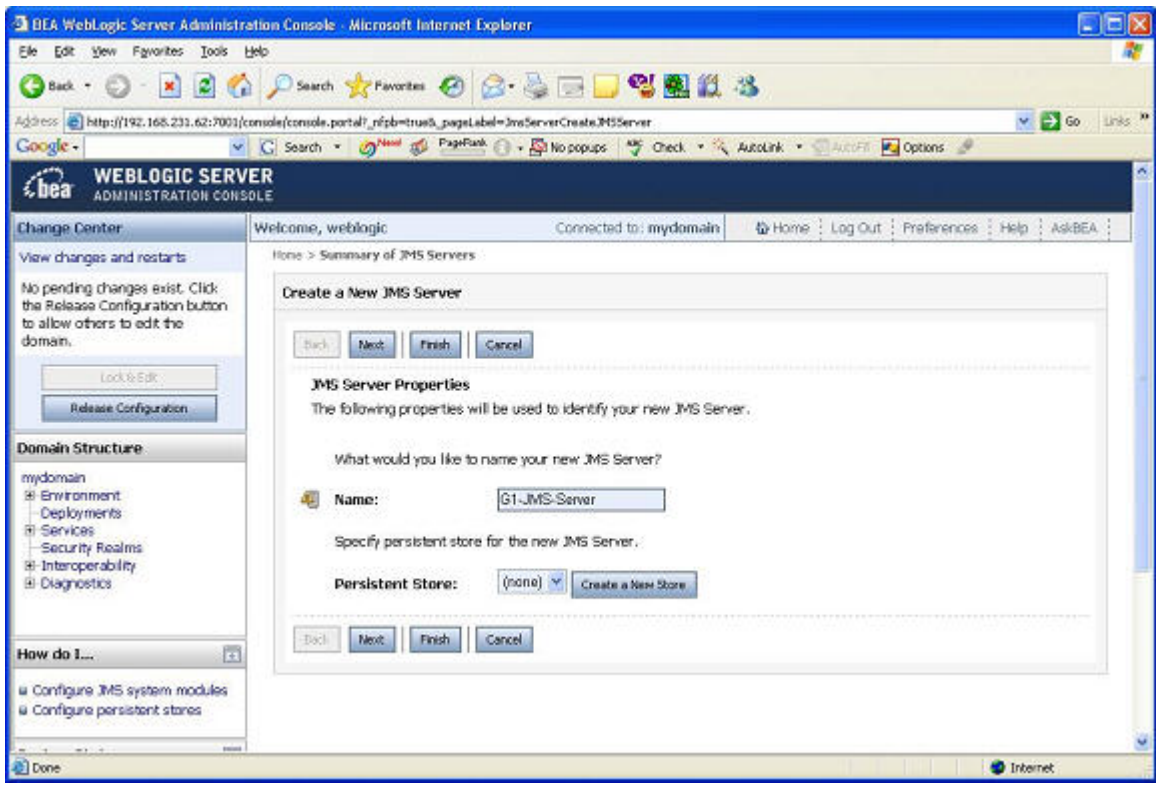
Spectrum™ Technology Platform Client API requires two JAR files to be present in the application: g1client.jar and g1message.jar. Both of them are stored in APP-INF/lib directory in clientapi.ear archive.

Deploying Client API EJB Application in WebLogic Server

Before deployment, the deployer must ensure that the associated JMS destination exists. For our message-driven bean implementation configuration, the server should have a JMS queue deployed with JNDI name MessageQueue. If the JMS destination does not exist, it needs to be created.

Follow the steps below to create a JMS destination associated with JMS queue in WebLogic Server:

1. Configure the JMS file store using the WebLogic administration console. All JMS-administered objects such as destinations or connection factories exist within a JMS Server. Each WebLogic Server instance can include multiple JMS Server instances. The following figure shows a JMS server named G1-JMS-Server that does not use any persistent store for storing JMS messages.



   After you have successfully created the JMS server, you will see a confirmation page.

2. JMS resources (such as JMS destinations, queues, topics, and so on) are contained in JMS modules that are used to configure and store configuration information, like normal J2EE modules. Create JMS module and call it G1-Module.
3. Add JMS resources to the module. To use message-driven bean we need to add JMS destination to our module; leave the **Would you like to add resources** checkbox checked.
4. Create JMS destinations. The application looks up the destination using JNDI; create destination "G1 JMS Queue" and set the JNDI name to "jms/G1QueueDestinationName".

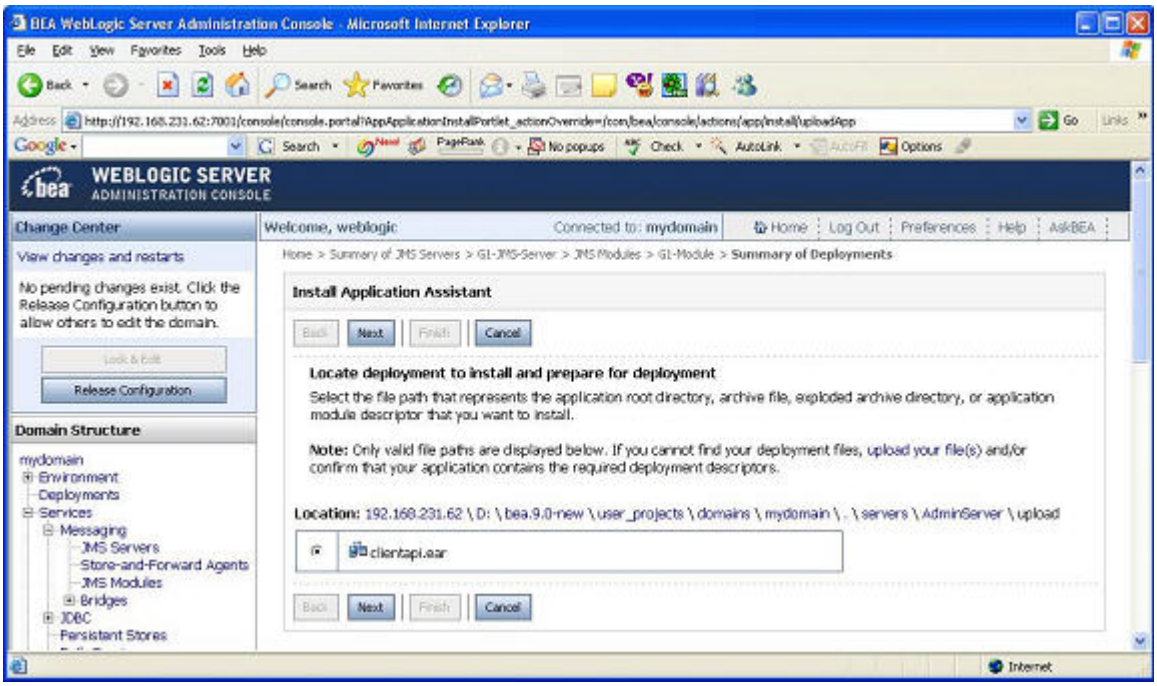
   After you create the JMS Destination, you will see the confirmation screen; the "Summary of Resources" contains our JMS Destination.

After building the EAR file and configuring the JMS destination to be used by Client API message-driven bean, you need to deploy the EAR on WebLogic Server.

5. Go to the WebLogic administrative console and select Domain Configuration > Applications > Your Deployed Resources.
6. Click Deploy New Application.

   Note: Before deploying an application it should be uploaded to WebLogic Server. Select upload your file(s) link to upload the application. When the upload dialog box appears, go to the location of clientapi.ear and upload the file.

   The following figure shows the uploaded file (clientapi.ear) ready for deployment:


BEA WebLogic Server Administration Console - Microsoft Internet Explorer
WEBLOGIC SERVER
ADMINISTRATION CONSOLE
Change Center
View changes and restarts
No pending changes exist. Click the Release Configuration button to allow others to edit the domain.
Release Configuration
Domain Structure
mydomain
Environment
Deployments
Services
Messaging
JMS Servers
Store-and-Forward Agents
JMS Modules
Bridges
JDBC
Persistent Stores
Welcome, weblogic
Connected to: mydomain
Home
Log Out
Preferences
Help
AskBEA
Home > Summary of JMS Servers > G1-JMS-Server > JMS Modules > G1-Module > Summary of Deployments
Install Application Assistant
Back
Next
Finish
Cancel
Locate deployment to install and prepare for deployment
Select the file path that represents the application root directory, archive file, exploded archive directory, or application module descriptor that you want to install.
Note: Only valid file paths are displayed below. If you cannot find your deployment files, upload your file(s) and/or confirm that your application contains the required deployment descriptors.
Location: 192.168.231.62 \ D: \ bea.9.0-new \ user_projects \ domains \ mydomain \ . \ servers \ AdminServer \ upload
clientapi.ear
Back
Next
Finish
Cancel
Internet

7. Select clientapi.ear and click Continue. You will see a screen that shows the deployment parameters.
8. Select Install this deployment as an application and click Next. You will see a summary of deployment. If there are any errors during deployment they will be shown on this page.
9. After successfully deploying an application, you need to start it. Select clientapi application and click Start.

   The next step you should take is to run the client web application for the Spectrum™ Technology Platform Client API (EJB).

10. Open the browser and enter the following URL: http://localhost:7001/clientapi/. If you are running the WebLogic Server on localhost and it listens to HTTP requests on port 7001, you should see a configuration page.
11. Enter the following information:

    - service name
    - column name for the request
    - value for that column

12. Click Submit. You will be taken to the page that displays results of processing the service.

Client API EJB for WebSphere 5.1.2

Packaging Spectrum™ Technology Platform Client API EJB

Before you can use the Client API EJB within WebSphere application server, it needs to be packaged and deployed in accordance with WebSphere requirements.

1. You should have already compiled class files for stateless session bean and message-driven bean implementations of Client API EJB and Java classes for Web application.
2. Import your reference EAR into WebSphere Studio Application Developer IDE.
3. Add WebSphere-specific deployment descriptor details.
4. Generate deployment for EAR.
5. After deployment is generated, you will test our Client API EJB inside test instance of WebSphere Application Server.

Loading Reference EAR Into WebSphere

To generate deployment unit for Client API EJB, you need to import your reference EAR into WebSphere Studio Application Developer IDE (WSAD).

1. Open WebSphere Studio Application Developer IDE (make sure you have version 5.1.2 or later installed so that it provides support for JDK 1.4 class format).
2. Select File > Import.
3. Select EAR file and click Next.
4. In the next dialog box, select the clientapi.ear file in the websphere.5.1.2\target directory of your Client API EJB installation.
5. Select WebSphere Application Server 5.1 as the target serve and click Next.
6. In the next dialog box, Import Defaults, click Next without changing anything.
7. Repeat step 6 for the next dialog box.
8. In the Manifest Class-Path dialog box, check all three dependent JARs for clientapi.jar and none for the rest of entries in the right-hand list ("JARs and module files in EAR").
9. Click Finish. You will see a new project in the Enterprise Applications folder in the right-hand window as well new entries in Web Modules and EJB Modules folders corresponding to clinetapi.war and clientapi.jar respectively. Your EAR has been imported into WASD IDE.
10. Ensure the Enterprise Application for Client API EJB is targeted for the correct WebSphere Application Server (WAS) version. You need targeting for version 5.1 to support JDK 1.4 class file format.
11. To make sure targeting is enabled, go to Window > Preferences and select J2EE in the tree control on the right. Inside the "Server Targeting Support" group ensure that "Enable Server Targeting Support" is enabled. If not, enable it and click Apply.
12. Close the Preferences dialog box and right-click on your enterprise application project in the right pane.

13. Select Target > ModifyÂ… The resulting dialog box contains list of application servers supported by WSAD.
14. Ensure WebSphere Application Server v5.1 is selected and that the checkbox below is checked; click OK. Your enterprise application project for Client API EJB will be updated with the targeting information.

Specifying Details for Application Server Deployment

Now you need to provide additional information for deploying the Client API EJB application on the WebSphere Application Server. Most of the deployment information is already specified in application server independent deployment descriptors, but some details (such JNDI name) for enterprise beans need to be specified in the application server specific way.

1. Open the EJB Modules folder.
2. Select your project and double-click on the Session Beans node. You should see "ClientApi" bean inside;
3. Double-click on "ClientApi"; the right pane will display the deployment information for that bean.
4. Under "WebSphere Bindings" heading you will see "JNDI name," which needs to be specified for other components to locate our stateless session bean via JNDI lookup. Set JNDI name to ejb/RemoteClientView/ClientApi.
5. Select File > Save.
6. Select ClientApiMDB in the list of beans; you will see configuration for message-driven bean.
7. Set ListenerPort name to jms/G1QueueDestinationName.
8. Select File > Save.

Generating Deployment Code

Before deploying the project to WAS, you must generate deployment code and configure JMS resources on WAS.

Right-click your enterprise application project and select Generate Deployment Code. After displaying a series of messages, deployment code will be generated. If there are any errors, they will be displayed in the lower pane of WSAD.

Configuring JMS Resources

To configure JMS resources, you need to see if you have a WebSphere testing instance set up. Open the Servers folder in the right pane and check that "WebSphere 5.1 Test Environment" is present. After double-clicking on that server you are going to see server configuration in the left pane with a number of tabs on the bottom of the pane.

1. Select the Configuration tab at the bottom of the screen and make sure the Enable administration console checkbox is checked. This will allow you to open the administrative console for that instance of WAS in the browser later.
2. Select the JMS tab to open the JMS configuration page. For this same application we will declare JMS resources at the Node level.
3. Scroll down to Node Settings section and expand it.
4. Next to the WASQueueConnectionFactory section, click Add.

5. In the Add WASQueueConnectionFactory dialog, enter the following values:
   - Name: G1JMSQueueConnectionFactory
   - JNDI Name: jms/ G1JMSQueueConnectionFactory
   - Node: select localhost
   - Server Name: select server1
6. Click OK to save the settings for this connection factory.
7. In the Node Settings section again, scroll down to the WASQueue entries section and click Add.
8. In the Add WASQueue dialog, enter the following values:
   - Name: G1JMSQueue
   - JNDI Name: jms/G1JMSQueue
   - Node: select localhost
9. Click OK to save the queue information.
10. Add information about listener ports for the message-driven bean. Listener ports are used to associate the JMS resources and deployed message-driven beans with the listener service that is handling the incoming messages.
    a) Select the EJB tab to open EJB configuration page for WAS.
    b) Click **Add**.
    c) In the Add ListenerPort dialog, enter the following values:
       - Name: jms/G1QueueDestinationName
       - Connection Factory JNDI name: select jms/G1JMSQueueConnectionFactory
       - Destination JNDI name: select jms/G1JMSQueue
    d) Click **OK** to save listener port information.
    e) Select File > Save to save configuration information for the server.

Running the Application

To run the enterprise application on test WAS instance, right-click on your enterprise application project for Client API then select Run on Server... You will see the server startup sequence in the console window.

To run client web application for Spectrum™ Technology Platform Client API (EJB), open the browser and enter the following URL:

http://localhost:9080/clientapi/

If you are running the WebSphere Server on localhost and it listens to HTTP requests on port 9080, you should see a configuration page.

Enter the following information:

- service name
- column name for the request
- value for that column

Client API EJB for WebSphere 6.0.1

Packaging Spectrum™ Technology Platform Client API EJB

Before you can use the Client API EJB within WebSphere application server, it needs to be packaged into an EAR file and deployed in accordance with WebSphere requirements.

1. You should have already compiled class files for stateless session bean and message-driven bean implementations of Client API EJB and Java classes for Web application.
2. Import your reference EAR into WebSphere JApplication Server using WebSphere Administrative Console.
3. After EAR is deployed, you will test your Client API EJB using Web Application contained in the reference EAR file.

Adding Class-Path to MANIFEST.MF

To allow EJB classes contained in clientapi.jar file to access Client API classes contained in g1message.jar and g1client.jar, as well as DOM4J classes contained in dom4j-1.6.1.jar, we need to make sure those utility JARs are specified in the classpath contained in Class-Path property inside MANIFEST.MF for clientapi.jar. Our Ant build file (build.xml) contains the following fragment to make sure Class-Path property is generated properly:

```
<manifest>
  <attribute name="Class-Path"
   value="APP-INF/lib/g1client.jar APP-INF/lib/g1message.jar APP-
INF/lib/dom4j-1.6.1.jar"/>
  <attribute name="Built-By" value="Group 1 Software"/>
</manifest>
```

This fragment is contained in "jar" target inside <jar> element.

Creating JMS Resources in WebSphere Application Server

Before deploying reference EAR into WebSphere Application Server, you need to create JMS resources required for message-driven bean (MDB) implementation of Client API. To create JMS resources, start WebSphere Application Server instance and then open WebSphere Administrative Console.

You will use V5 Default Messaging Provider for JMS services.

1. Open the Resources group in the right pane of the administrative console.
2. Open the JMS Providers group and select V5 default messaging.

   You will see the V5 default messaging provider configuration page.
3. In the provider configuration page, select WebSphere queue connection factories to open the connection factories configuration page for that provider, then click New.
4. Create a new connection factory with the following parameters:

   - Scope: select Node
   - Name: G1JMSQCF

- JNDI name: jms/G1JMSQCF

5. Save the configuration. Be sure you actually save it since it involves multiple steps in IBM WebSphere Administrative console.
6. Create a queue destination for the default provider.
   a) Return to provider configuration page and select WebSphere queue destinations.
   b) Click New.
   c) Create a new queue destination with the following parameters:
      - Scope: select Node
      - Name: G1JMSQueueDestination
      - JNDI name: jms/G1JMSQueueDestination
7. Save the configuration.

   After the configuration is saved, you need to create a Listener Port to be used by the message-driven bean.
8. Go to your server configuration page by opening the Servers group in the left pane, then selecting Application servers and selecting your server from the list.
9. Scroll down to Communications on the server configuration page and open the Messaging group.
10. Click Message Listener Service to get to the message listener configuration page for that server.
11. Select Listener Ports and then click New to create a new listener port.
12. Enter the following information:
    - Name: G1MDBListenerPort
    - Connection factory JNDI name: jms/G1JMSQCF
    - Destination JNDI name: jms/G1JMSQueueDestination
13. Click Apply to save the configuration. Then proceed to start all JMS resources that were created.

Deploying the Reference EAR File

1. Run "ant clean ear" in your build directory for WebSphere 6.0.1.
2. Open Applications in the left pane of the administrative console and select Install New Application.
3. Select the location of reference EAR file for Client API EJB by using the Browse button.
4. Click Next. Leave all defaults and click Next again. You will see a page with a sequence of steps for deployment.
5. Retain all defaults on Step 1 and click Next.
6. Step 2 will show that reference EAR contains two applications: one that contains EJBs and the other that is used as a client application to issue requests and display results. Leave all defaults and click Next.
7. Retain all defaults on Step 3 and click Next.
8. Select Listener port option on Step 4 and specify "G1MDBListenerPort" for the name of the port. This will allow the message-driven bean to use the listener port we have created earlier.
9. Specify "ejb/RemoteClientView/ClientApi" on Step 5 as the JNDI name for the ClientApi bean and click Next.
10. Retain all defaults on Step 6 and click Next.
11. Retain all defaults on Step 7 and click **Next**.

12. Click Finish on Step 8 to start the deployment. The next page will show the deployment progress and any errors that were detected during deployment:

    Toward the bottom of the page you'll see "Application Spectrum™ Technology Platform Client API installed successfully." The application has been deployed on the server.

    Now you need to save the deployment. Select Save to Master Configuration and then click Save on the next page. You should see list of enterprise applications that contains "Spectrum™ Technology Platform Client API"; the application should be in "running" state (green arrow).

Running the Application

To run client web application for Spectrum™ Technology Platform Client API (EJB), open the browser and enter the following URL:

http://localhost:9080/clientapi/

If you are running the WebSphere Server on localhost and it listens to HTTP requests on port 9080, you should see a configuration page:

Enter the following information:

- service name
- column name for the request
- value for that column

4

The C API

In this section:

- Introduction to the C API 58
- Server 71
- Service 75
- Message 76
- DataTable 88
- DataRow 96

Introduction to the C API

The C API consists of the following structures:

- Server
- Service
- Message
- DataTable
- DataRow

Note: The C API is a C wrapper around the C++ code. On Unix you can use a C++ compiler to build your C application, which is the preferred approach. However, a C compiler can also be used directly on Linux, and Solaris. On HP-UX and AIX, you need to link all the C++ required libs when you use the C compiler. To do this, run "ldd ./batch" under ..../samples/batch/bin/ to get the list of all dependent libs and put them in the link section of your makefile.

Supported Libraries

Spectrum™ Technology Platform provides an ASCII and Unicode version C API, while the Unicode version remains as compatible as possible with the original ASCII-version API design. Spectrum™ Technology Platform applies International Components for Unicode (ICU) in the API to support the Unicode feature. ICU is a mature, widely used set of C/C++ libraries for Unicode support and is developed by IBM.

The Unicode standard defines a default encoding based on 16-bit code units. This is supported in ICU by the definition of the UChar to be an unsigned 16-bit integer type(unsigned short *). This is the base type for character arrays for strings in ICU. Spectrum™ Technology Platform uses UChar as the Unicode string representation in our C API.

Note: Not all services support the full Unicode character set. For example, the ValidateAddress service supports the ISO 8859-1 character set for US inputs and International inputs and outputs and the CP 850 character set for Canadian inputs and outputs. However, the Unicode libraries should be used whenever your input data may contain any non-ASCII character, even if the underlying service does not support the full Unicode character set.

For detailed information about UChar, please refer to the following two sites:

- icu.sourceforge.net/userguide/
- www-306.ibm.com/software/globalization/icu/index.jsp

C Libraries Supported on Windows

Each API configuration produces library files with a common base name (g1client) but with a unique suffix and possibly prefix ("lib" in the case of static libraries). The library suffixes work like this:

```
<lib>g1client<S><U><D>.<lib|dll>
```

- lib—indicates a static library.

- dll—indiates a dynamic (shared) library.
- S—indicates a single-threaded build. If this is absent it indicates a multi-threaded build.
- U—indicates a UNICODE version build. If this is absent it indicates an ASCII build.
- D—indicates a debug build. If this suffix is absent it indicates an optimized release build.

To enable the UNICODE version, the LIB_UNICODE macro definition must be in your project.

To use the static C/C++ API library UNICODE version, you need to define U_STATIC_IMPLEMENTATION in your project.

To use the dynamic version, you need to define G1CLIENT_DLL in your project.

We also provide a file called "auto_link.h" in the header file directory and it automatically links to all the corresponding libraries according to the project settings.

To call 64-bit libraries in Windows, you need to define VER_64 in your project.

Static Library

Note: The names provided in this section are for 32-bit libraries. For 64-bit libraries, replace "32" with "64" in the library name.

Table 7: Single Threaded/Release

	ASCII	Unicode
g1	libg1client_S.lib	libg1client_SU.lib
openssl	otlibeay32.lib otlibssl32.lib	otlibeay32.lib otlibssl32.lib
opentop	opentop.lib	opentopw.lib
icu		libicuuc.lib libicudt.lib libicuin.lib libicuio.lib
Poco	PocoXML32.lib	PocoXML32w.lib

Table 8: Single Threaded/Debug

	ASCII	Unicode
g1	libg1client_SD.lib	libg1client_SUD.lib
openssl	otlibeay32d.lib otlibssl32d.lib	otlibeay32d.lib otlibssl32d.lib
opentop	opentopd.lib	opentopwd.lib

	ASCII	Unicode
icu		libicuucd.lib libicudtd.lib libicuind.lib libicuiod.lib
Poco	PocoXML32d.lib	PocoXML32wd.lib

Table 9: Multi/Release (using Multi-Threaded CRT)

	ASCII	Unicode
g1	libg1client.lib	libg1client_U.lib
openssl	otlibeay32mt.lib otlibssl32mt.lib	otlibeay32mt.lib otlibssl32mt.lib
opentop	opentopmt.lib	opentopmtw.lib
icu		libicuucmt.lib libicudtmt.lib libicuinmt.lib libicuiomt.lib
Poco	PocoXMLmt32.lib	PocoXML32mtw.lib

Table 10: Multi/Debug (using Multi-Threaded CRT)

	ASCII	Unicode
g1	libg1client_D.lib	libg1client_UD.lib
openssl	otlibeay32mtd.lib otlibssl32mtd.lib	otlibeay32mtd.lib otlibssl32mtd.lib
opentop	opentopmtd.lib	opentopmtwd.lib
icu		libicuucmtd.lib libicudtmtd.lib libicuinmtd.lib libicuiomtd.lib
Poco	PocoXMLmt32d.lib	PocoXML32mtwd.lib

Dynamic Library

Note: The names provided in this section are for 32-bit libraries. For 64-bit libraries, replace "32" with "64" in the library name.

Table 11: Multi/Release (using Multi-Threaded CRT)

	ASCII	Unicode
g1	g1client.dll	g1client_U.dll
openssl	otlibeay32mts.dll otlibssl32mts.dll	otlibeay32mts.dll otlibssl32mts.dll
opentop	opentopmts.dll	opentopmtws.dll
icu		icuuc32.dll icuio32.dll icuin32.dll icudt32.dll
Poco	PocoXML32mts.dll	PocoXML32mtws.dll

Table 12: Multi/Debug (using Multi-Threaded CRT)

	ASCII	Unicode
g1	g1client_D.dll	g1client_UD.dll
openssl	otlibeay32mtds.dll otlibssl32mtds.dll	otlibeay32mtds.dll otlibssl32mtds.dll
opentop	opentopmtds.dll	opentopmtwds.dll
icu		icuuc32d.dll icuio32d.dll icuin32d.dll icudt32d.dll
Poco	PocoXML32mtds.dll	PocoXML32mtwds.dll

C Libraries Supported on Unix

Each ClientSDK configuration produces library files with a common base name (libg1client) but with a unique suffix. Spectrum™ Technology Platform provides a multithread and release build for ASCII version and UNICODE version.

The library suffixes work like this:-

```
libg1client<U>.<so|sl|a>
```

- U—indicates a UNICODE version build. If this is absent it indicates an ASCII build.

To use the UNICODE version, you need to define LIB_UNICODE in your project.

In UNICODE Version C++ API, the namespace for all classes is g1client.

Table 13: AIX

	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libssl.so	libcrypto.so libssl.so
opentop	libopentop-xlCmt.so	libopentop-xlCmtw.so libotxml-xlCmtw.so
icu		libicudata34.a libicui18n34.a libicuio34.a libicuuc34.a
Poco	libPocoXML.so	

Table 14: HP-UX

	ASCII	Unicode
g1	libg1client.sl	libg1client_U.sl
openssl	libcrypto.sl libssl.sl libcrypto.sl.0.9.7 libssl.sl.0.9.7	libcrypto.sl libssl.sl libcrypto.sl.0.9.7 libssl.sl.0.9.7
opentop	libopentop-accmt.sl	libopentop-accmtw.sl libotxml-accmtw.sl
icu		libicudata.sl libicudata.sl.34 libicui18n.sl libicui18n.sl.34 libicuio.sl libicuio.sl.34 libicuuc.sl libicuuc.sl.34
Poco	libPocoXML.sl	

Table 15: Itanium

	ASCII	Unicode
g1	libg1client.sl	libg1client_U.sl
openssl	libcrypto.a libssl.a	libcrypto.a

	ASCII	Unicode
		libssl.a
opentop	libopentop-accmt.sl	libopentop-accmtw.sl libotxml-accmtw.sl
icu		libicudata.sl libicudata.sl.34 libicudata.sl.34.0 libicui18n.sl libicui18n.sl.34 libicui18n.sl.34.0 libicuio.sl libicuio.sl.34 libicuio.sl.34.0 libicuuc.sl libicuuc.sl.34 libicuuc.sl.34.0
Poco	libPocoXML.sl	

Table 16: Linux

	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7
opentop	libopentop-gccmt.so	libopentop-gccmtw.so libotxml-gccmtw.so
icu		libicudata.so libicudata.so.34 libicui18n.so libicui18n.so.34 libicuio.so libicuio.so.34 libicuuc.so libicuuc.so.34
Poco	libPocoXML.so	

Table 17: Solaris

	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7

	ASCII	Unicode
opentop	libopentop-fortemt.so	libopentop-fortemtw.so libotxml-fortemtw.so
icu		libicudata.so libicudata.so.34 libicui18n.so libicui18n.so.34 libicuio.so libicuio.so.34 libicuuc.so libicuuc.so.34
Poco	libPocoXML.so	

Constants

The C API uses two sets of constants. The first set is for the Server component, described in the table below.

Table 18: Constants for the Server Component

Constant Name	Description/Default	Example
SERVER_HOST	String for server host name. Default is "localhost".	65.89.200.89
SERVER_PORT	String for server port. Default is "8080".	10119
SERVER_ACCOUNT_ID	String for server account ID. No default value.	user1
SERVER_ACCOUNT_PASSWORD	String for server account password. No default value.	user1
SERVER_CONNECTION_TIMEOUT	String for server connection timeout, in milliseconds. Default is "5000".	50000
SERVER_CONNECTION_TYPE	String for server connection type. Currently only supports HTTP, HTTPS, or SOCKET. Default is "HTTP".	HTTP(S)

Constant Name	Description/Default	Example
SERVER_PROXY_HOST	String for proxy server host name. No default value.	192.168.1.77
SERVER_PROXY_PORT	String for proxy server port. No default value.	8080
SERVER_PROXY_USER	String for proxy server account ID. No default value.	user1
SERVER_PROXY_PASSWORD	String for proxy server account password. No default value.	user1

The second set of constants is for the Message component.

Table 19: Constants for the Message Component

Constant Name	Description	Example
MESSAGE_CONTEXT_ACCOUNT_ID	String for message context account ID.	user1
MESSAGE_CONTEXT_ACCOUNT_PASSWORD	String for message context account password.	user1
MESSAGE_CONTEXT_SERVICE_NAME	String for message context service name.	echoservice

Error Messages

Some functions return a SUCCESSFUL_RETURN or 0 (zero) value if they are successful. If it is not successful, the function returns the corresponding error code. In order to retrieve error messages, call getErrorMessage(int errorCode). For example:

```
Server *server = NULL;
int nRet;
//Create Server
server = createServer();
//set the property to the server
...
//Connect to server
printf("Making connection to the server...\n");
nRet = serverConnect(server);
if(nRet != SUCCESSFUL_RETURN)
{
// ASCII Version-use the following code
```

```
printf(getErrorMessage(nRet));
//Unicode Version -use the following code
UChar * error = getErrorMessage(nRet));
// more code to print out the error messageÂ….
 return ;
 }
```

The C API uses the following error messages:

- Error Messages for Passing Null structure:
- "Input null DataRow"
- "Input null DataTable"
- "Input null Message"
- "Input null Server"
- Error Messages for Connection
- "Connection type not supported"
- "Client timeout"
- "Blank connection property name"
- "Blank property name"
- Error Messages for Creating DataTable:
- "Blank column name"
- "Duplicated column name"
- Error Messages for MessagePackaging Exception
- "Input Message is null"
- "Failed to connect to Server"
- "Failed to disconnect from Server"
- "Failed to open Http Connection"
- "Failed to get Service"
- "Failed to package the message using Serializer and Encoding"

Example Application

The sample code shown below illustrates how to use the ASCII version C API.

```
// Declarations
Server *server = NULL;
Message *request = NULL;
DataTable *dataTable = NULL;
Message *reply = NULL;
Service *service = NULL;
int nRet;
DataRow *row1 = NULL;
DataRow *row2 = NULL;
DataTable *returnDataTable= NULL;
char** columnNames;
DataRow** rows;
DataRow*dataRow;
int i;
int j;
char* value;

//Create Server
```

```
server = createServer();

//Set server connection properties
nRet = setConnectionProperty(server, SERVER_HOST, "localhost");
nRet = setConnectionProperty(server, SERVER_PORT, "10119 ");
nRet = setConnectionProperty(server, SERVER_CONNECTION_TYPE,"SOCKET");
nRet = setConnectionProperty(server, SERVER_ACCOUNT_ID,"guest");
nRet = setConnectionProperty(server, SERVER_ACCOUNT_PASSWORD,"");

//Connect to server
nRet = serverConnect(server);
if(nRet != SUCCESSFUL_RETURN)
{
printf( getErrorMessage(nRet));
// free memory
if(server)
nRet = deleteServer(server);
return ;
}

 //Get Service From Server
 service = getServiceFromServer(server,"ValidateAddress" );

//Create Input Message
request = createMessage();

//Fill DataTable in the input message
dataTable = getDataTable(request);
nRet= addColumn( dataTable, "AddressLine1", &nRet);
nRet= addColumn( dataTable, "City", &nRet);
nRet= addColumn( dataTable, "StateProvince", &nRet);

row1 = newRow( dataTable );
setByIndex (row1, 0 , "4200 Parliament Place");
setByIndex (row1, 1 ,"Lanham");
setByIndex (row1, 2 , "Maryland");

addRow( dataTable, row1);

row2 = newRow( dataTable );
setByIndex (row2, 0 , "10535 Boyer Blvd");
setByIndex (row2, 1 ,"Austin");
setByIndex (row2, 2 , "Texas");

addRow( dataTable, row2);

//Set"option" Properties to the Input Message
nRet = putOption(request, "OutputCasing","M");
nRet = putOption(request, "OutputRecordType","A");

//Process Input Message, return output Message
nRet = processMessage(service, request, &reply);
if(nRet != SUCCESSFUL_RETURN)
{
printf("Error Occurred, " );
printf(getErrorMessage(nRet));

// free memory
if(request)
nRet = deleteMessage(request);
if(reply)
```

```
nRet = deleteMessage(reply);
if(server)
nRet = deleteServer(server);

return ;

}

//Disconnect from server
nRet = serverDisconnect(server);

//Get the result from the response message
returnDataTable = getDataTable(reply );
columnNames = getColumnNames(returnDataTable);

rows = getDataRows( returnDataTable);

for( i=0; i < getRowCount( returnDataTable); i++)
{
dataRow = rows[i];

for(j=0; j < getColumnCount(returnDataTable); j++)
{
value = (char*)getByIndex( dataRow, j);
printf(value);
printf("\n");
}
}

//Free Memory
if(request)
nRet = deleteMessage(request);
if(reply)
nRet = deleteMessage(reply);
if(server)
nRet = deleteServer(server);
}
```

The sample code shown below illustrates how to use the Unicode version C API. The string here is represented by UChar*(or unsigned short*), which is 16-bit type to represent the Unicode string .ICU provides a function called u_charsToUChars, which converts 8-bit string to 16-bit string. The example here shows how to call Unicode version C API. The input string are all ASCII, so that we use u_charsToUChars to convert to 16-bit string. You could also construct Unicode string to directly pass in C API.

```
UChar* convertcharToUChar( char* name, UChar* value)
  {
   int lenName= strlen(name);
   u_charsToUChars(name, value, lenName );
   value[ lenName]=0;
   return value;
  }

   // Declarations
   Server *server = NULL;
   Message *request = NULL;
   DataTable *dataTable = NULL;
   DataTable *returnDataTable= NULL;
   Message *reply = NULL;
   Service *service = NULL;
```

```
   int nRet;
   DataRow* newDataRow;
   UChar    name[128];
   UChar value[128];
   UChar** columnNames;
   DataRow** rows;
   DataRow* dataRow;
   int i, j;
   UChar* columnValue;
   UChar*   errorMsg;

   //Create Server
   server = createServer();

   //Set server connection properties
   setConnectionProperty(server, convertcharToUChar( SERVER_HOST, name) ,
convertcharToUChar( "localhost", value));
   setConnectionProperty(server, convertcharToUChar( SERVER_PORT, name) ,
convertcharToUChar( "10119", value));
  setConnectionProperty(server, convertcharToUChar( SERVER_CONNECTION_TYPE,
 name) , convertcharToUChar( "SOCKET", value));
  setConnectionProperty(server, convertcharToUChar( SERVER_ACCOUNT_ID, name)
 , convertcharToUChar( "guest", value));
  setConnectionProperty(server, convertcharToUChar( SERVER_ACCOUNT_PASSWORD,
 name) , convertcharToUChar( "", value));


   //Connect to server
   nRet = serverConnect(server);
   if(nRet != SUCCESSFUL_RETURN)
   {
   // error handling
   errorMsg = getErrorMessage(nRet);
   // free memory
   if(server)
   nRet = deleteServer(server);
   return ;
   }

   //Get Service From Server
  service = getServiceFromServer(server,convertcharToUChar( "ValidateAddress",
 name));

   //Create Input Message
   request = createMessage();

   //Fill DataTable in the input message
   dataTable = getDataTable(request);
   addColumn( dataTable,convertcharToUChar( "AddressLine1", name),  &nRet);

   addColumn( dataTable,convertcharToUChar( "City", name),  &nRet);
   addColumn( dataTable,convertcharToUChar( "PostalCode", name),  &nRet);
  addColumn( dataTable,convertcharToUChar( "StateProvince", name),  &nRet);


   newDataRow = newRow( dataTable );

   setByIndex (newDataRow, 0 , convertcharToUChar( "74, Rue Octave Bénard",
 name) );
   setByIndex (newDataRow, 1 , convertcharToUChar( "Etang-Salé-les- Bains",
 name) );
```

```
   setByIndex (newDataRow, 2 , convertcharToUChar( "97427", name) );
   setByIndex (newDataRow, 3 , convertcharToUChar( "Reunion Island", name)
);

   addRow( dataTable, newDataRow);

   //Set"option" Properties to the Input Message
   nRet = putOption(request, convertcharToUChar( "OutputCasing", name),
convertcharToUChar( "M", value));
   nRet = putOption(request, convertcharToUChar( "OutputRecordType", name),
 convertcharToUChar( "A", value));


   //Process Input Message, return output Message
   nRet = processMessage(service, request, &reply);
    if(nRet != SUCCESSFUL_RETURN)
   {
   // error handling
   errorMsg = getErrorMessage(nRet);
   // free memory
   if(request)
   nRet = deleteMessage(request);
   if(reply)
   nRet = deleteMessage(reply);
   if(server)
   nRet = deleteServer(server);

   return ;
   }

   //Disconnect from server
   nRet = serverDisconnect(server);

   //Get the result from the response message
   returnDataTable = getDataTable(reply );
   columnNames = getColumnNames(returnDataTable);
   rows = getDataRows( dataTable);
   for( i=0; i < getRowCount( dataTable); i++)
   {
   dataRow =  rows[i];
   for(j=0; j < getColumnCount(dataTable); j++)
   {
   columnValue = (UChar*)getByIndex( dataRow, j);
   }
   }

   //Free Memory
   if(request)
   nRet = deleteMessage(request);
   if(reply)
   nRet = deleteMessage(reply);
   if(server)
   nRet = deleteServer(server);
```

Server

The Server structure is used to connect to server, disconnect from the server, and get the service from the server. The following table summarizes the functions performed in the Server structure.

Table 20: Server Functions Summary

Function	Description
createServer	Creates a server.
deleteServer	Deletes the server.
setConnectionProperty	Sets the configuration items for the connection to the server.
serverConnect	Connects to the server.
serverDisconnect	Disconnects from the server.
getServiceFromServer	Gets the service from the server.

Note: See the **Modules** on page 241 section of this guide for a list of services that may be available to you.

CreateServer

Creates the server.

Syntax

```
Server* createServer()
```

Parameters

None.

Result

The server is created.

Example

```
Server *server = NULL;
//Create Server
server = createServer();
```

DeleteServer

Deletes the server.

Syntax

```
int deleteServer(Server* server)
```

Parameters

- Server— the server to be deleted.

Result

Returns 0 (if successful) or error code.

Example

```
int nRet;
nRet = deleteServer(server);
```

SetConnectionProperty

Establishes the server connection configuration properties, such as host name and length of timeout.

Syntax

ASCII Version

```
int setConnectionProperty(Server* server, const char* name, const char* value)
```

Unicode Version

```
int setConnectionProperty(Server* server, const UChar* name, const UChar*
value)
```

Parameters

- Server — the server to which the client connects
- Name — the name of the connection property, such as HOST
- Value — the value for the name of the connection property, such as "www.myhost.com"

Result

Returns 0 (if successful) or error code.

Example

ASCII Version

```
int nRet;
Server *server = NULL;
nRet = createServer(&server);
nRet = setConnectionProperty(server, SERVER_HOST,
"localhost");
```

Unicode Version

```
int nRet;
// construct 16-bit string
UChar serverHost[32];
char* SERVER_HOST= SERVER_HOST;
u_charsToUChars(SERVER_HOST, serverHost, strlen(SERVER_HOST));
serverHost [ strlen(SERVER_HOST)]=0;
// construct 16-bit string
UChar hostValue [32];
char* value= "localhost";
u_charsToUChars(value, hostValue, strlen(value));
hostValue[ strlen(value)]=0;
nRet = setConnectionProperty(server, serverHost , hostValue);
```

ServerConnect

Reads the properties to determine the configuration settings and makes a connection to the server.

Note: C uses the HTTP, HTTPS, or SOCKET server connection protocol. HTTP and HTTPS logically establish a client connection but do not actually connect to the server until a GetService or Process method is invoked. The SOCKET protocol establishes a connection to the server when Connect is invoked.

Syntax

```
int serverConnect(Server* server)
```

Parameters

- Server—the server to which the client connects

Results

Returns 0 (if successful) or error code.

Example

```
int nRet;
nRet = serverConnect(server);
```

ServerDisconnect

Disconnects from the server.

Syntax

```
int serverDisconnect(Server* server)
```

Parameters

- Server—the server from which the client disconnects.

Results

Returns 0 (if successful) or error code.

Example

```
int nRet;
nRet = serverDisconnect(server);
```

GetServiceFromServer

Gets the service from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Syntax

ASCII Version

```
Service* getServiceFromServer(Server* server,const char* serviceName )
```

Unicode Version

```
Service* getServiceFromServer(Server* server,const UChar* serviceName )
```

Parameters

- Server - server from which the client connects
- ServiceName - the name of service the client requests

Results

Service returned.

Example

ASCII Version

```
Server *server= NULL;
Service *service = NULL;
//Create Server
server = createServer();
...
// get Service From Server
service = getServiceFromServer(server,"ValidateAddress" );
```

Unicode Version

```
// construct 16-bit string
UChar serviceName[32];
char* sName="ValidateAddress";
u_charsToUChars(sName, serviceName, strlen(sName));
serviceName [ strlen(sName)]=0;
service = getServiceFromServer(server , serviceName );
```

Service

The Service structure is used to process the message (in other words, it sends the message to the server and receives a response from the server). The following table summarizes the functions performed in the Service structure.

Table 21: Service Functions Summary

processMessage	Processes the input message and retrieves the response message from the server.

ProcessMessage

Process the input message and returns the response message.

Note: You will need to call DeleteMessage() to free memory when this returned message is no longer used.

Syntax

```
int processMessage (Service* service, Message* request, Message* returnVal)
```

Parameters

- Service—the service the client requests.
- Request—the input message which contains the "option" setting and the dataset.
- returnVal—returns the response message from the server.

Results

Returns 0 (if successful) or error code.

Example

```
Message *request = NULL;
Message *reply = NULL;
int nRet;
 ...
```

```
 // Assume that service is given here
 // Create Input Message
 request = createMessage();
 ... more code to fill dataTable information in request message
//Process Input Message, return output Message
 nRet = processMessage(service, request, &reply);
 if(nRet != SUCCESSFUL_RETURN)
 {
 printf("Error Occurred, " );
 printf(getErrorMessage(nRet));
 return ;
 }
 if(request)
 nRet = deleteMessage(request);
 if(reply)
 nRet = deleteMessage(reply);
```

Message

The Message structure sends your input data and receives your output data from the service. The properties for Message include context properties, such as account ID, account password, service name, and service method; option properties, which are the service-specific runtime options; and error properties, which are the error class, error message and error stacktrace.

The following table summarizes the functions performed in the Message structure.

Table 22: Message Functions Summary

Function	Description
createMessage	Creates a message.
deleteMessage	Deletes the message.
getContext	Gets the value of the context entity identified by the name in the context session of the message.
getContextMap	Gets the Map that contains all of the context entries.
putContext	Sets the value of the context entity identified by the name in the context session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putContextMap	Adds the new context properties to the current context properties.

Function	Description
setContextMap	Overwrites the current context properties with the new context properties.
getOption	Gets the value of the option entity identified by the name in the option session of the message.
getOptions	Gets the Map that contains all of the option entries.
putOption	Sets the value of the option entity identified by the name in the option session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putOptions	Adds the new option properties to the current option properties.
setOptions	Overwrites the current option properties with the new option properties.
getError	Gets the error message.
getDataTable	Gets the DataTable from the message

These methods are discussed in more detail in the following sections.

CreateMessage

Creates a message.

Syntax

```
Message*      createMessage()
```

Parameters

None.

Results

The message created.

Example

```
Message* request = NULL;
request = createMessage();
```

DeleteMessage

Deletes the message.

Syntax

```
int deleteMessage(Message* message)
```

Parameters

- Message— the message to be deleted

Results

Returns 0 if successful or error code.

Example

```
int nRet = deleteMessage(message);
```

GetContext

Gets the value of the context entity identified by the name in the context session of the message. "Context" entities include the following constants: account id, account password, service name, service method.

Syntax

ASCII Version

```
const char* getContext(Message* message,const char* name)
```

Unicode Version

```
const UChar * getContext(Message* message,const UChar* name)
```

Parameters

- Message - the message to which this function applies
- Name - the name whose associated value is to be returned

Result

Returns the value for the name in the context entity. If the name does not exist, the method returns empty string.

Example

ASCII Version

```
const char* value = getContext(message, "account.id");
```

Unicode Version

```
UChar* value;
// construct 16-bit string
UChar accountID[32];
char* account="account.id";
u_charsToUChars(account, accountID, strlen(account));
accountID[ strlen(account)]=0;
value = getContext(message, accountID);
```

GetContextMap

Gets the Map that contains all of the context entries.

Syntax

ASCII Version

```
MAP_STRING**getContextMap(Message* message)
Where the MAP_STRING is defined by
typedef struct map_string{
char* key;
char* value;
}MAP_STRING;
```

Unicode Version

```
MAP_STRING**getContextMap(Message* message)
Where the MAP_STRING is defined by
typedef struct map_string{
UChar* key;
UChar* value;
}MAP_STRING;
```

Parameters

- Message - the message to which this function applies

Results

Returns the array of MAP_STRING that contains all of the context entries.

Example

ASCII Version

```
int i;
char* name;
char* value;
MAP_STRING** mapping;
mapping = getContextMap( message);
i=0;
while(mapping[i] != NULL)
{
name= mapping[i]->key;
value = mapping[i]->value;
i++;
}
```

Unicode Version

```
int i;
UChar* name;
UChar* value;
MAP_STRING** mapping;
mapping = getContextMap( message);
i=0;
while(mapping[i] != NULL)
{
name= mapping[i]->key;
value = mapping[i]->value;
i++;
}
```

PutContext

Sets the value for given name in the "context" properties. If there is an existing value present for the entity identified by the name, it is replaced. Context properties include the following constants: account ID, account password, service name, service key, and request ID.

Syntax

ASCII Version

```
int putContext(Message* message,const char* name,
const char* value)
```

Unicode Version

```
int putContext(Message* message,const UChar* name,
const UChar* value)
```

Parameters

- Message—message to which this function applies
- Name—name with which the specified value is to be associated
- Value—value to be associated with the specified name

Results

Returns 0 (if successful) or error code.

Example

ASCII Version

```
int nRet;
Message* message = createMessage();
nRet = putContext( message, "account.id", "user1") ;
```

Unicode Version

```
int nRet;
Message* message;
// construct 16-bit string
UChar accountID[32];
char* account="account.id";
```

```
UChar accountIDValue[32];
char* accountValue="user1";
u_charsToUChars(account, accountID, strlen(account));
accountID [ strlen(account)]=0;
u_charsToUChars(accountValue, accountIDValue, strlen(accountValue));
accountIDValue [ strlen(accountValue)]=0;
message = createMessage();
nRet = putContext( message, accountID, accountIDValue);
```

PutContextMap

Adds the new context properties to the current context properties.

Syntax

```
int putContextMap(Message* message, MAP_STRING** context)
```

Parameters

- Message - the message to which this function applies
- The new context map to be added to the current context map.

Results

Returns 0 (if successful) or error code.

Example

ASCII Version

```
MAP_STRING** mapping;
Message* message;
message = createMessage();
int nRet;
mapping = (MAP_STRING **)malloc(3 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = "key1" ;
mapping[0]->value = "value1" ;
mapping[1] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[1]->key = "key2" ;
mapping[1]->value = "value2" ;
mapping[2] = NULL;
nRet = putContextMap( message, mapping) ;
```

Unicode Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
UChar key1[32];
char* key1String="key1";
UChar value1[32];
char* value1String="value1";

u_charsToUChars(key1String, key1, strlen(key1String));
key1[ strlen(key1String)]=0;
u_charsToUChars(value1String, value1, strlen(value1String));
value1[ strlen(value1String)]=0;
```

```
message = createMessage();
mapping = (MAP_STRING **)malloc(2 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = key1;
mapping[0]->value = value1 ;
mapping[1] = NULL;
nRet = putContextMap( message, mapping) ;
```

SetContextMap

Overwrites the current context properties with the new context properties.

Syntax

```
int setContextMap(Message* message, MAP_STRING** context)
```

Parameters

- Message - the message to which this function applies
- The new context map to be used to replace the current context map.

Results

Returns 0 (if successful) or error code.

Example

ASCII Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
message = createMessage();
mapping = (MAP_STRING **)malloc(2 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = "key1" ;
mapping[0]->value = "value1" ;
mapping[1] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[1]->key = "key2" ;
mapping[1]->value = "value2" ;
mapping[2] = NULL;
nRet=setContextMap( message, mapping) ;
```

Unicode Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
UChar key1[32];
char* key1String="key1";
UChar value1[32];
char* value1String="value1";
u_charsToUChars(key1String, key1, strlen(key1String));
key1[ strlen(key1String)]=0;
u_charsToUChars(value1String, value1, strlen(value1String));
value1[ strlen(value1String)]=0;
message = createMessage();
```

```
mapping = (MAP_STRING **)malloc(2 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = key1 ;
mapping[0]->value = value1 ;
mapping[1] = NULL;
nRet=setContextMap( message, mapping) ;
```

GetOption

Gets the value of the option entity identified by the name in the option session of the message. "Options" entities include the service-specific runtime options, such as output casing, output data format, and so on.

Syntax

ASCII Version

```
const char* getOption(Message* message,const char* name)
```

Unicode Version

```
const Uchar* getOption(Message* message,const UChar* name)
```

Parameters

- Message - the message to which this function applies
- Name- the name whose associated value is to be returned

Results

Returns the value for the name in the "option" property in the message or an empty string if the name does not exist.

Example

ASCII Version

```
const char* value = getOption (message, " OutputCasing");
```

Unicode Version

```
UChar* value;
// construct 16-bit string
UChar option[32];
char* optionValue="OutputCasing";
u_charsToUChars(optionValue, option, strlen(optionValue));
option [ strlen(optionValue)]=0;
value = getOption(message, option);
```

GetOptions

Gets the map that contains all of the option entries.

Syntax

```
MAP_STRING**        getOptions(Message* message)
```

Parameters

- Message—the message to which this function applies

Results

Returns the array of MAP_STRING that contains all of the context entries.

Example

ASCII Version

```
int i;
char* name;
char* value;
MAP_STRING** mapping;
mapping = getOptions( message);
i=0;
while(mapping[i] != NULL)
{
name= mapping[i]->key;
value = mapping[i]->value;
i++;
}
```

Unicode Version

```
int i;
UChar* name;
UChar* value;
MAP_STRING** mapping;
mapping = getOptions( message);
i=0;
while(mapping[i] != NULL)
{
name= mapping[i]->key;
value = mapping[i]->value;
i++;
}
```

PutOption

Sets the value for given name in the "option" properties. If there is an existing value present for the entity identified by the name, it is replaced. Option properties are the service-specific run-time options.

Syntax

ASCII Version

```
int putOption(Message* message,const char* name,
const char* value)
```

Unicode Version

```
int putOption(Message* message,const UChar* name,
const UChar* value)
```

Parameters

- Message - the message to which this function applies
- Name - with which the specified value is to be associated.
- Value - to be associated with the specified name.

Results

Returns 0 (if successful) or error code.

Example

ASCII Version

```
int nRet;
Message* message = createMessage();
nRet = putOption( message, "OutputCasing", "M");
```

Unicode Version

```
int nRet;
Message* message;
// construct 16-bit string
UChar option[32];
char* optionString="OutputCasing";

UChar optionValue[32];
char* optionValueString="M";

u_charsToUChars(optionString, option, strlen(optionString));
option[ strlen(optionString)]=0;

u_charsToUChars(optionValueString, optionValue,
strlen(optionValueString));
optionValue [ strlen(optionValueString)]=0;

message = createMessage();
nRet = putOption( message, option, optionValue);
```

PutOptions

Adds the new option properties to the current option properties.

Syntax

```
int  putOptions(Message* message, MAP_STRING** context)
```

Parameters

- Message - the message to which this function applies
- The new option map to be added to the current option properties

Results

Returns 0 if successful or error code.

Example

ASCII Version

```
MAP_STRING** mapping;
Message* message;
message = createMessage();
int nRet;
mapping = (MAP_STRING **)malloc(3 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = "key1" ;
mapping[0]->value = "value1" ;
mapping[1] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[1]->key = "key2" ;
mapping[1]->value = "value2" ;
mapping[2] = NULL;
nRet = putOptions( message, mapping) ;
```

Unicode Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
UChar key1[32];
char* key1String="key1";
UChar value1[32];
char* value1String="value1";
u_charsToUChars(key1String, key1, strlen(key1String));
key1[ strlen(key1String)]=0;
u_charsToUChars(value1String, value1, strlen(value1String));
value1[ strlen(value1String)]=0;
message = createMessage();
mapping = (MAP_STRING **)malloc(2 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = key1;
mapping[0]->value = value1 ;
mapping[1] = NULL;
nRet = putOptions ( message, mapping) ;
```

SetOptions

Overwrites the current option properties with the new option properties.

Syntax

```
int  setOptions(Message* message, MAP_STRING** context)
```

Parameters

- Message - the message to which this function applies
- The new option map to be used to replace the current option map

Results

Returns 0 if successful or error code.

Example

ASCII Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
message = createMessage();
mapping = (MAP_STRING **)malloc(3 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = "key1" ;
mapping[0]->value = "value1" ;
mapping[1] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[1]->key = "key2" ;
mapping[1]->value = "value2" ;
mapping[2] = NULL;
nRet=setOptions( message, mapping) ;
```

Unicode Version

```
MAP_STRING** mapping;
Message* message;
int nRet;
UChar key1[32];
char* key1String="key1";
UChar value1[32];
char* value1String="value1";
u_charsToUChars(key1String, key1, strlen(key1String));
key1[ strlen(key1String)]=0;
u_charsToUChars(value1String, value1, strlen(value1String));
value1[ strlen(value1String)]=0;
message = createMessage();
mapping = (MAP_STRING **)malloc(2 * sizeof(MAP_STRING *));
mapping[0] = (MAP_STRING *)malloc( sizeof(MAP_STRING));
mapping[0]->key = key1 ;
mapping[0]->value = value1 ;
mapping[1] = NULL;
nRet= setOptions ( message, mapping) ;
```

GetError

Gets the error message from the message.

Syntax

ASCII Version

```
const char* getError(Message* message )
```

Unicode Version

```
const UChar* getError(Message* message )
```

Parameters

- Message - the message to which this function applies

Result

Returns the error message in message.

Example

ASCII Version

```
const char* error = getError(message );
```

Unicode Version

```
const UChar* error = getError(message );
```

GetDataTable

Gets the DataTable in the message.

Syntax

```
DataTable*   getDataTable(Message* message )
```

Parameters

- Message - the message to which this function applies

Example

```
// Assume that message is given here
DataTable  *dataTable   ;
dataTable = getDataTable( message );
```

DataTable

DataTable contains the records for the input and output data.

The following table summarizes the functions performed in the DataTable structure.

Table 23: DataTable Functions Summary

Function	Description
createDataTable	Creates the DataTable.
deleteDataTable	Deletes the DataTable.
addColumn	Adds the new column.
getColumnNames	Gets all the column names.

Function	Description
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
clear	Clears the data in DataTable.
getDataRows	Gets an array of all DataRows in the DataTable.
addRow	Adds a DataRow to the DataTable.
newRow	Creates a new DataRow in the DataTable.
getRowCount	Gets the number of the DataRows in this DataTable.
mergeDataTable	Merges the given DataTable and the current DataTable.

These methods are discussed in more detail in the following sections.

CreateDataTable

Create the DataTable.

Syntax

```
DataTable* createDataTable()
```

Results

Returns the DataTable created.

Example

```
DataTable* dataTable;
dataTable = createDataTable();
```

DeleteDataTable

Deletes the DataTable.

Syntax

```
int deleteDataTable(DataTable* dataTable)
```

Parameters

- Datatable - the DataTable to be deleted

Example

```
DataTable* dataTable;
dataTable = createDataTable();
...
if(dataTable) deleteDataTable(dataTable);
```

AddColumn

Adds the new column.

Syntax

ASCII Version

```
int addColumn(DataTable* dataTable, const char* columnName,
int* indexReturn)
```

Unicode Version

```
int addColumn(DataTable* dataTable, const UChar* columnName,
int* indexReturn)
```

Parameters

- Datatable - the DataTable to which this function applies
- Column name to be added to the DataTable
- Returns the corresponding index

Results

Returns 0 if successful or error code.

Exceptions

- Blank column name
- Duplicate column name

Example

ASCII Version

```
int nIndex;
int nRet;
nRet= addColumn( dataTable, "AddressLine1", &nIndex);
nRet= addColumn( dataTable, "City",&nIndex);
nRet= addColumn( dataTable, "State", &nIndex);
if(nRet != SUCCESSFUL_RETURN)
{
printf(getErrorMessage(nRet));
return ;
}
```

Unicode Version

```
int nRet;
int nIndex;
UChar* error;
UChar city[64];
char* cityString= "City"
u_charsToUChars(cityString, city, strlen(cityString));
city[ strlen(cityString)]=0;

nRet= addColumn( dataTable, city,&nIndex);
if(nRet != SUCCESSFUL_RETURN)
{
error = getErrorMessage(nRet);
//more code
}
```

GetColumnNames

Gets all the column names.

Syntax

ASCII Version

```
char** getColumnNames(dataTable )
```

Unicode Version

```
UChar** getColumnNames(dataTable )
```

Parameters

- Datatable - the DataTable to which this function applies

Results

Returns the array of column names.

Example

ASCII Version

```
char* value;
char** columnNames;
int i;
columnNames =getColumnNames ( dataTable) ;
for( i=0; i < getColumncount( dataTable); i++)
{
value = columnNames[i];
}
```

Unicode Version

```
UChar* value;
UChar** columnNames;
int i;
columnNames =getColumnNames ( dataTable) ;
for( i=0; i < getColumncount( dataTable); i++)
```

```
{
value = columnNames[i];
}
```

GetColumnIndex

Gets the corresponding column index.

Syntax

ASCII Version

```
int getColumnIndex(DataTable* dataTable ,const char* columnName)
```

Unicode Version

```
int getColumnIndex(DataTable* dataTable ,const UChar* columnName)
```

Parameters

- Datatable - the DataTable to which this function applies
- Column name

Results

Returns the corresponding column index.

Example

ASCII Version

```
int nIndex ;
nIndex = getColumnIndex(dataTable ,"AddressLine1")
```

Unicode Version

```
int nIndex ;
UChar columnName[64];
char* columnNameStr= "AddressLine1" u_charsToUChars(columnNameStr, columnName,
 strlen(columnNameStr));
columnName [strlen(columnNameStr)]=0;
nIndex = getColumnIndex(dataTable , columnName);
```

GetColumnCount

Gets the number of columns.

Syntax

```
int getColumnCount(DataTable* dataTable )
```

Parameters

- Datatable - the DataTable to which this function applies

Results

Returns the number of columns.

Example

```
// Assume that dataTable is given here int nColumnCount ;
nColumnCount = getColumnCount( dataTable ) ;
```

Clear

Clears the data in DataTable.

Syntax

```
int clear(DataTable* dataTable)
```

Parameters

- Datatable - the DataTable to which this function applies

Results

Returns 0 if successful or error code.

Example

```
// Assume that dataTable is given here
clear(dataTable);
```

GetDataRows

Gets an array of all DataRows in the DataTable.

Syntax

```
DataRow** getDataRows(DataTable* dataTable)
```

Parameters

- Datatable - the DataTable to which this function applies

Results

Returns an array of DataRows.

Example

```
// Assume that dataTable is given here
DataRows** rows;
DataRow* dataRow;
int i;
int j;
```

```
rows = getDataRows( dataTable);
for( i=0; i < getRowCount( dataTable); i++)
{
 dataRow =  rows[i];

 for(j=0; j < getColumnCount(dataTable); j++)
 {
 value = (char*)getByIndex( dataRow, j);
 }
}
```

AddRow

Adds a DataRow to the DataTable.

Syntax

```
int addRow(DataTable* dataTable, DataRow* dataRow)
```

Parameter

- Datatable - the DataTable to which this function applies
- Datarow to be added to the DataTable

Results

Returns 0 if successful or error code.

Example

```
// Assume that dataTable is given here DataRow* newDataRow;
int nRet;
newDataRow = newRow( dataTable );
setByIndex (newDataRow, 0 , "10535 Boyer Blvd");
setByIndex (newDataRow, 1 , "Austin");
setByIndex (newDataRow, 2 , "Texas");
nRet = addRow( dataTable, newDataRow);
```

NewRow

Creates a new DataRow in the DataTable.

Syntax

```
DataRow* newRow(DataTable* dataTable )
```

Parameter

- Datatable - the DataTable to which this function applies

Results

Returns the new created DataRow.

Example

```
// Assume that dataTable is given here
DataRow* newDataRow;
int nRet;
newDataRow = newRow( dataTable );
setByIndex (newDataRow, 0 , "10535 Boyer Blvd");
setByIndex (newDataRow, 1 , "Austin");
setByIndex (newDataRow, 2 , "Texas");
nRet = addRow( dataTable, newDataRow);
```

GetRowCount

Gets the number of the DataRows in this DataTable.

Syntax

```
int getRowCount(DataTable* dataTable)
```

Parameter

- Datatable - the DataTable to which this function applies

Results

Returns the number of the DataRows in this DataTable.

Example

```
// Assume that dataTable is given here int nRowCount ;
nRowCount = getRowCount( dataTable);
```

MergeDataTable

Merges the given DataTable and the current DataTable.

Syntax

```
int mergeDataTable(DataTable* dataTable ,DataTable* other )
```

Parameter

- Datatable - the DataTable to which this function applies
- Other DataTable to be merged with the current DataTable

Results

Returns 0 if successful or error code.

Example

```
// Assume that dataTable and otherDataTable are given here
mergeDataTable (dataTable ,otherDataTableDataRow)
```

DataRow

DataRow contains the record for the input and output data.

The following table summarizes the functions performed in the DataRow structure.

Table 24: DataRow Functions Summary

Function	Description
createDataRow	Creates the DataRow.
deleteDataRow	Deletes the DataRow.
getColumnNamesFromRow	Gets all the column names.
getColumnIndexFromRow	Gets the corresponding column index
getColumnCountFromRow	Gets the number of columns.
getByIndex	Gets the value from the fields array by the column index in this DataRow.
getByName	Gets the value from the fields array by the column name in this DataRow.
mergeDataRow	Merges the given DataRow and the current DataRow.
setByName	Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.
setByIndex	Sets the value for the corresponding column for the DataRow. If the value for the index exists, the old value is replaced.
addChild	Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow Collection, otherwise a new Collection will be created with the supplied DataRow as its only element.

Function	Description
getChildren	Retrieves the child rows from a named relationship.
listChildNames	Retrieves all of the names of the named parent/child relationships.
setChildren	Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

CreateDataRow

Creates the DataRow.

Syntax

```
DataRow*   createDataRow()
```

Results

Returns the DataRow created.

Example

```
DataRow* dataRow;
dataRow = createDataRow();
```

DeleteDataRow

Deletes the DataRow.

Syntax

```
int   deleteDataRow(DataRow* dataRow)
```

Parameter

- The DataRow to be deleted

Example

```
DataRow* dataRow;
dataRow = createDataRow();
...
if(dataRow)
  deleteDataRow (dataRow);
```

GetColumnNamesFromRow

Gets all the column names.

Syntax

ASCII Version

```
char** getColumnNamesFromRow(DataRow* dataRow)
```

Unicode Version

```
UChar** getColumnNamesFromRow(DataRow* dataRow)
```

Parameter

- Datarow - the DataRow to which this function applies

Results

Returns the array of column names.

Example

ASCII Version

```
char* value;
char** columnNames;
int i;
columnNames = getColumnNamesFromRow (dataRow) ;
for( i=0; i < getColumnCountFromRow (dataRow); i++)
{
 value = columnNames[i];
}
```

Unicode Version

```
UChar* value;
UChar** columnNames;
int i;
columnNames = getColumnNamesFromRow (dataRow) ;
for( i=0; i < getColumnCountFromRow (dataRow); i++)
{
 value = columnNames[i];
}
```

GetColumnIndexFromRow

Gets the corresponding column index.

Syntax

ASCII Version

```
int getColumnIndexFromRow(DataRow* dataRow, const char* name)
```

Unicode Version

```
int getColumnIndexFromRow(DataRow* dataRow, const UChar* name)
```

Parameter

- Datarow - the DataRow to which this function applies
- Column name

Results

Returns the corresponding column index.

Example

ASCII Version

```
int nIndex
nIndex = getColumnIndexFromRow ("AddressLine1");
```

Unicode Version

```
int nIndex
UChar columnName[64];
char* columnNameStr= "AddressLine1"
u_charsToUChars(columnNameStr, columnName, strlen(columnNameStr));
columnName [strlen(columnNameStr)]=0;
nIndex = getColumnIndexFromRow (columnName);
```

GetColumnCountFromRow

Gets the number of columns.

Syntax

```
int  getColumnCountFromRow(DataRow* dataRow )
```

Parameter

- Datarow - the DataRow to which this function applies

Results

Returns the number of columns.

Example

```
//Assume that the dataRow is given here
int nColumnCount ;
nColumnCount = getColumnCountFromRow (dataRow );
```

GetByIndex

Gets the value from the fields array by the column index in this DataRow.

Syntax

ASCII Version

```
const char* getByIndex(DataRow* dataRow, int index)
```

Unicode Version

```
const UChar* getByIndex(DataRow* dataRow, int index)
```

Parameter

- Datarow - the DataRow to which this function applies
- Index with which the specified value is to be associated.

Results

Returns the value for the column index in the DataRow, returns empty string if the index in invalid.

Example

ASCII Version

```
char* value = getByIndex( dataRow, 0);
```

Unicode Version

```
UChar* value = getByIndex( dataRow, 0);
```

GetByName

Gets the value from the fields array by the column name in this DataRow.

Syntax

ASCII Version

```
const char* getByName(DataRow* dataRow, const char* name )
```

Unicode Version

```
const UChar* getByName(DataRow* dataRow, const UChar* name )
```

Parameter

- Datarow - the DataRow to which this function applies
- Name with which the specified value is to be associated

Results

Returns the value for the column name in the DataRow, returns empty string if the column name does not exist.

Example

ASCII Version

```
char* value = getByName ( dataRow, "City")
```

Unicode Version

```
UChar* value;
UChar columnName[64];
char* columnNameStr= "City"
u_charsToUChars(columnNameStr, columnName, strlen(columnNameStr));
columnName [strlen(columnNameStr)]=0;
value = getByName ( dataRow, columnName);
```

MergeDataRow

Merges the given DataRow and the current DataRow.

Syntax

```
int   mergeDataRow(DataRow* dataRow, DataRow* other)
```

Parameter

- Datarow - the DataRow to which this function applies
- Other DataRow to be merged with the current DataRow

Results

Returns 0 if successful or error code.

Example

```
//Assume that the dataRow and otherDataRow are given here
int nRet;
nRet= mergeDataRow(dataRow, otherDataRow);
```

SetByName

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

ASCII Version

```
int setByName(DataRow* dataRow, const char* name, const char* value)
```

Unicode Version

```
int setByName(DataRow* dataRow, const UChar* name, const
UChar* value)
```

Parameters

- Datarow - the DataRow to which this function applies
- Name with which the specified value is to be associated
- Value to be associated with the specified name

Exceptions

If input Blank column name or Duplicate column name, return error

Results

Returns 0 if successful or error code.

Example

ASCII Version

```
int nRet;
nRet= setByName (dataRow, "City", "Austin");
if(nRet != SUCCESSFUL_RETURN)
{ printf(getErrorMessage(nRet));
//more code
}
```

Unicode Version

```
int nRet;
UChar* error;
UChar columnName[64];
char* columnNameStr= "City"
UChar columnValue[64];
char* columnValueStr= "Austin";
u_charsToUChars(columnNameStr, columnName, strlen(columnNameStr));
columnName [strlen(columnNameStr)]=0;
u_charsToUChars(columnValueStr, columnValue, strlen(columnValueStr));
columnValue [strlen(columnValueStr)]=0;
nRet= setByName (dataRow, columnName, columnValue);
if(nRet != SUCCESSFUL_RETURN)
{ error = getErrorMessage(nRet);
//more code
}
```

SetByIndex

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

ASCII Version

```
int setByIndex(DataRow* dataRow, int index, const char* value)
```

Unicode Version

```
int setByIndex(DataRow* dataRow, int index, const UChar* value)
```

Parameters

- Datarow - the DataRow to which this function applies
- Column index with which the specified value is to be associated
- Value to be associated with the specified name

Exceptions

- The column index is invalid

Results

Returns 0 if successful or error code.

Example

ASCII Version

```
int nRet;
nRet= setByIndex (dataRow, 1, "Austin");
if(nRet != SUCCESSFUL_RETURN)
{
printf(getErrorMessage(nRet));
 //more code
}
```

Unicode Version

```
int nRet;
UChar* error;
UChar columnValue[64];
char* columnValueStr= "Austin";
u_charsToUChars(columnValueStr, columnValue, strlen(columnValueStr));
columnValue [strlen(columnValueStr)]=0;
nRet= setByIndex (dataRow, 1, columnValue);
if(nRet != SUCCESSFUL_RETURN)
{
error = getErrorMessage(nRet);
//more code
}
```

AddChild

Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow collection. Otherwise, a new collection will be created with the supplied DataRow as its only element.

Syntax

ASCII Version

```
void addChild(DataRow* dataRow, const char* childName, DataRow* childDataRow)
```

Unicode Version

```
void addChild(DataRow* dataRow, const UChar* childName, DataRow* childDataRow)
```

Parameters

- The name of the parent/child relationship (e.g., "Flood Plain Data," "References," "Used By," and so forth)
- The DataRow to be added to the relationship

Example

ASCII Version

```
DataRow* dataRow = createDataRow();
DataRow* child1DataRow1 = createDataRow();

setByName(child1DataRow1, "City", "Austin");
setByName(child1DataRow1, "State", "Texas");

addChild( dataRow, "child1", child1DataRow1);
```

Unicode Version

```
UChar* convertcharToUChar( char* name, UChar* value)
 {
 int lenName= strlen(name);

 u_charsToUChars(name, value, lenName );

 value[ lenName]=0;
 return value;
 } >
DataRow* dataRow = createDataRow();
DataRow* child1DataRow1 = createDataRow();
UChar     name[128];
UChar     columnValue[128];
setByName(child1DataRow1, convertcharToUChar("City", name),
  convertcharToUChar("Austin", columnValue));
setByName(child1DataRow1, convertcharToUChar("State", name),
  convertcharToUChar("Texas", columnValue));
addChild( dataRow, "child1", child1DataRow1);
```

GetChildren

Retrieves the child rows from a named relationship.

Syntax

ASCII Version

```
DataRow** getChildren(DataRow* dataRow,const char* childName)
```

Unicode Version

```
DataRow** getChildren(DataRow* dataRow,const UChar* childName)
```

Parameters

- The name of the parent/child relationship, for example "Flood Plain Data", "References", "Used By", and so forth.

Results

Returns the child rows from the named relationship.

Example

ASCII Version

```
DataRow** child1Rows;
child1Rows = getChildren(dataRow, "child1");
```

Unicode Version

```
DataRow** child1Rows;
UChar childName[128];
/* see convertcharToUChar in the Example section of "addChild" */
child1Rows = getChildren(dataRow, convertcharToUChar("child1", childName));
```

ListChildNames

Retrieves all of the names of the named parent/child relationships.

Syntax

ASCII Version

```
char** listChildNames(DataRow* dataRow)
```

Unicode Version

```
UChar** listChildNames(DataRow* dataRow)
```

Results

Returns the set of the names of the named parent/child relationships.

Example

ASCII Version

```
char** childsNames;
childsNames =listChildNames( dataRow);
```

Unicode Version

```
UChar** childsNames;
childsNames=listChildNames( dataRow);
```

SetChildren

Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

Syntax

ASCII Version

```
DataRow** setChildren(DataRow* dataRow, const char* childName, DataRow**
dataRows)
```

Unicode Version

```
DataRow** setChildren(DataRow* dataRow, const UChar* childName, DataRow**
dataRows)
```

Results

Returns the set of the names of the named parent/child relationships.

Example

ASCII Version

```
DataRow* dataRow = createDataRow();
DataRow* child1DataRow1 = createDataRow();
DataRow* child1DataRow2 = createDataRow();
DataRow* child2DataRow = createDataRow();
DataRow** child2Rows;
DataRow** returnRows;

setByName(child1DataRow1, "Address", "200 Congress");
setByName(child1DataRow1, "City", "Austin");

setByName(child1DataRow2, "Address", "100 Congress");
setByName(child1DataRow2, "City", "Dallas");

setByName(child2DataRow, "Address", "100 Congress");
setByName(child2DataRow, "City", "Austin");

addChild( dataRow, "child1", child1DataRow1);
addChild( dataRow, "child1", child1DataRow2);
addChild( dataRow, "child2", child2DataRow );

child2Rows=getChildren(dataRow, "child2");

returnRows=setChildren( dataRow, "child1", child2Rows);
```

Unicode Version

```
DataRow* dataRow = createDataRow();
DataRow* child1DataRow1 = createDataRow();
DataRow* child1DataRow2 = createDataRow();
DataRow* child2DataRow = createDataRow();
DataRow** child2Rows;
DataRow** returnRows;
UChar name[128];
UChar columnValue[128];
```

```
UChar childName[128];

setByName(child1DataRow1, convertcharToUChar("Address", name),
convertcharToUChar("200 Congress", columnValue));
setByName(child1DataRow1, convertcharToUChar("City", name),
convertcharToUChar("Austin", columnValue));
setByName(child1DataRow2, convertcharToUChar("Address", name),
convertcharToUChar("100 Congress", columnValue));
setByName(child1DataRow2, convertcharToUChar("City", name)
convertcharToUChar("Dallas", columnValue) );
setByName(child2DataRow, convertcharToUChar("Address", name),
convertcharToUChar("100 Congress", columnValue) );
setByName(child2DataRow, convertcharToUChar("City", name),
convertcharToUChar("Austin", columnValue) );

addChild( dataRow, convertcharToUChar("child1", childName), child1DataRow1);
addChild( dataRow, convertcharToUChar("child1", childName),child1DataRow2);
addChild( dataRow, convertcharToUChar("child2", childName), child2DataRow );

child2Rows=getChildren(dataRow, convertcharToUChar("child2",childName));
returnRows=setChildren( dataRow, convertcharToUChar("child1",childName),
child2Rows);
```

5

The C++ API

In this section:

- Introduction to the C++ API 110
- Server 121
- Service 124
- Message 125
- DataTable 133
- DataRow 139

Introduction to the C++ API

The C++ API consists of the following classes:

- Server
- Service
- Message
- DataTable
- DataRow

UnicodeString in ICU is a string class that stores Unicode characters directly and provides similar functionality as the Java String and StringBuffer classes. The Spectrum™ Technology Platform Unicode C++ API uses this class to store Unicode strings.

Supported Libraries

Spectrum™ Technology Platform provides an ASCII and Unicode version C API, while the Unicode version remains as compatible as possible with the original ASCII-version API design. Spectrum™ Technology Platform applies International Components for Unicode (ICU) in the API to support the Unicode feature. ICU is a mature, widely used set of C/C++ libraries for Unicode support and is developed by IBM.

The Unicode standard defines a default encoding based on 16-bit code units. This is supported in ICU by the definition of the UChar to be an unsigned 16-bit integer type(unsigned short *). This is the base type for character arrays for strings in ICU. Spectrum™ Technology Platform uses UChar as the Unicode string representation in our C API.

Note: Not all services support the full Unicode character set. For example, the ValidateAddress service supports the ISO 8859-1 character set for US inputs and International inputs and outputs and the CP 850 character set for Canadian inputs and outputs. However, the Unicode libraries should be used whenever your input data may contain any non-ASCII character, even if the underlying service does not support the full Unicode character set.

For detailed information about UChar, please refer to the following two sites:

- icu.sourceforge.net/userguide/
- www-306.ibm.com/software/globalization/icu/index.jsp

Windows

Each API configuration produces library files with a common base name (g1client) but with a unique suffix and possibly prefix ("lib" in the case of static libraries). The library suffixes work like this:

```
<lib>g1client<S><U><D>.<lib|dll>
```

- lib—indicates a static library.
- dll—indiates a dynamic (shared) library.
- S—indicates a single-threaded build. If this is absent it indicates a multi-threaded build.
- U—indicates a UNICODE version build. If this is absent it indicates an ASCII build.

- D—indicates a debug build. If this suffix is absent it indicates an optimized release build.

To enable the UNICODE version, the LIB_UNICODE macro definition must be in your project.

To use the static C/C++ API library UNICODE version, you need to define U_STATIC_IMPLEMENTATION in your project.

To use the dynamic version, you need to define G1CLIENT_DLL in your project.

We also provide a file called "auto_link.h" in the header file directory and it automatically links to all the corresponding libraries according to the project settings.

To call 64-bit libraries in Windows, you need to define VER_64 in your project.

Static Library

Note: The names provided in this section are for 32-bit libraries. For 64-bit libraries, replace "32" with "64" in the library name.

Single Threaded/Release		
	ASCII	Unicode
g1	libg1client_S.lib	libg1client_SU.lib
openssl	otlibeay32.lib otlibssl32.lib	otlibeay32.lib otlibssl32.lib
opentop	opentop.lib	opentopw.lib
icu		libicuuc.lib libicudt.lib libicuin.lib libicuio.lib
Poco	PocoXML32.lib	PocoXML32w.lib
Single Threaded/Debug		
	ASCII	Unicode
g1	libg1client_SD.lib	libg1client_SUD.lib
openssl	otlibeay32d.lib otlibssl32d.lib	otlibeay32d.lib otlibssl32d.lib
opentop	opentopd.lib	opentopwd.lib
icu		libicuucd.lib libicudtd.lib libicuind.lib libicuiod.lib
Poco	PocoXML32d.lib	PocoXML32wd.lib

Multi/Release (using Multi-Threaded CRT)		
	ASCII	Unicode
g1	libg1client.lib	libg1client_U.lib
openssl	otlibeay32mt.lib otlibssl32mt.lib	otlibeay32mt.lib otlibssl32mt.lib
opentop	opentopmt.lib	opentopmtw.lib
icu		libicuucmt.lib libicudtmt.lib libicuinmt.lib libicuiomt.lib
Poco	PocoXMLmt32.lib	PocoXML32mtw.lib
Multi/Debug (using Multi-Threaded CRT)		
	ASCII	Unicode
g1	libg1client_D.lib	libg1client_UD.lib
openssl	otlibeay32mtd.lib otlibssl32mtd.lib	otlibeay32mtd.lib otlibssl32mtd.lib
opentop	opentopmtd.lib	opentopmtwd.lib
icu		libicuucmtd.lib libicudtmtd.lib libicuinmtd.lib libicuiomtd.lib
Poco	PocoXMLmt32d.lib	PocoXML32mtwd.lib

Dynamic Library

Note: The names provided in this section are for 32-bit libraries. For 64-bit libraries, replace "32" with "64" in the library name.

Multi/Release (using Multi-Threaded CRT)		
	ASCII	Unicode
g1	g1client.dll	g1client_U.dll
openssl	otlibeay32mts.dll otlibssl32mts.dll	otlibeay32mts.dll otlibssl32mts.dll

opentop	opentopmts.dll	opentopmtws.dll
icu		icuuc32.dll icuio32.dll icuin32.dll icudt32.dll
Poco	PocoXML32mts.dll	PocoXML32mtws.dll
Multi/Debug (using Multi-Threaded CRT)		
	ASCII	Unicode
g1	g1client_D.dll	g1client_UD.dll
openssl	otlibeay32mtds.dll otlibssl32mtds.dll	otlibeay32mtds.dll otlibssl32mtds.dll
opentop	opentopmtds.dll	opentopmtwds.dll
icu		icuuc32d.dll icuio32d.dll icuin32d.dll icudt32d.dll
Poco	PocoXML32mtds.dll	PocoXML32mtwds.dll

Unix

Each ClientSDK configuration produces library files with a common base name (libg1client) but with a unique suffix. Spectrum™ Technology Platform provides a multithread and release build for ASCII version and UNICODE version.

The library suffixes work like this:-

```
libg1client<U>.<so|sl|a>
```

- U—indicates a UNICODE version build. If this is absent it indicates an ASCII build.

To use the UNICODE version, you need to define LIB_UNICODE in your project.

In UNICODE Version C++ API, the namespace for all classes is g1client.

AIX		
	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libssl.so	libcrypto.so libssl.so

opentop	libopentop-xlCmt.so	libopentop-xlCmtw.so libotxml-xlCmtw.so
icu		libicudata34.a libicui18n34.a libicuio34.a libicuuc34.a
Poco	libPocoXML.so	
HP-UX		
	ASCII	Unicode
g1	libg1client.sl	libg1client_U.sl
openssl	libcrypto.sl libssl.sl libcrypto.sl.0.9.7 libssl.sl.0.9.7	libcrypto.sl libssl.sl libcrypto.sl.0.9.7 libssl.sl.0.9.7
opentop	libopentop-accmt.sl	libopentop-accmtw.sl libotxml-accmtw.sl
icu		libicudata.sl libicudata.sl.34 libicui18n.sl libicui18n.sl.34 libicuio.sl libicuio.sl.34 libicuuc.sl libicuuc.sl.34
Poco	libPocoXML.sl	
Itanium		
	ASCII	Unicode
g1	libg1client.sl	libg1client_U.sl
openssl	libcrypto.a libssl.a	libcrypto.a libssl.a
opentop	libopentop-accmt.sl	libopentop-accmtw.sl libotxml-accmtw.sl
icu		libicudata.sl libicudata.sl.34 libicudata.sl.34.0 libicui18n.sl libicui18n.sl.34 libicui18n.sl.34.0 libicuio.sl libicuio.sl.34

		libicuio.sl.34.0 libicuuc.sl libicuuc.sl.34 libicuuc.sl.34.0
Poco	libPocoXML.sl	
Linux		
	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7
opentop	libopentop-gccmt.so	libopentop-gccmtw.so libotxml-gccmtw.so
icu		libicudata.so libicudata.so.34 libicui18n.so libicui18n.so.34 libicuio.so libicuio.so.34 libicuuc.so libicuuc.so.34
Poco	libPocoXML.so	
Solaris		
	ASCII	Unicode
g1	libg1client.so	libg1client_U.so
openssl	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7	libcrypto.so libcrypto.so.0.9.7 libssl.so libssl.so.0.9.7
opentop	libopentop-fortemt.so	libopentop-fortemtw.so libotxml-fortemtw.so
icu		libicudata.so libicudata.so.34 libicui18n.so libicui18n.so.34 libicuio.so libicuio.so.34 libicuuc.so libicuuc.so.34
Poco	libPocoXML.so	

Constants

The C++ API uses two sets of constants. The first set is for the Server class described in the table below.

Table 25: Constants for the Server Component

Constant Name	Description/Default	Example
Server::HOST	String for server host name. Default is "localhost".	65.89.200.89
Server::PORT	String for server port. Default is "8080".	10119
Server::ACCOUNT_ID	String for server account ID. No default value.	user1
Server::ACCOUNT_PASSWORD	String for server account password. No default value.	user1
Server::CONNECTION_TIMEOUT	String for server connection timeout, in milliseconds. Default is "5000".	50000
Server::CONNECTION_TYPE	String for server connection type. Currently only supports HTTP, HTTPS, or SOCKET. Default is "HTTP".	HTTP(S)
Server::PROXY_HOST	String for proxy server host name. No default value.	192.168.1.77
Server::PROXY_PORT	String for proxy server port. No default value.	8080
Server::PROXY_USER	String for proxy server account ID. No default value.	user1
Server::PROXY_PASSWORD	String for proxy server account password. No default value.	user1

The second set of constants is for the Message class:

Table 26: Constants for the Message Component

Constant Name	Description	Example
Message::CONTEXT_ACCOUNT_ID	String for message context account ID.	user1
Message::CONTEXT_ACCOUNT_PASSWORD	String for message context account password.	user1
Message::CONTEXT_SERVICE_NAME	String for message context service name.	echoservice

Error Messages

In order to get error messages, use the Exception class. Use the try/catch constructs to capture the error message. For example:

```
try{
  Server *server=new Server();

  //Connect to server
  server->connect();

  }catch(Exception e)
  {
   // ASCII Version-use the following code
  cout << "Error Occurs," << e.getErrorMessage();
  //Unicode Version -use the following code

  UnicodeString error = e.getErrorMessage() ;

  wcout << error.getTerminatedBuffer();
  }
```

The C++ API uses the following error messages:

- Error Messages for Connection:
- "Connection type not supported"
- "Client timeout"
- "Blank connection property name"
- "Blank property name"
- Error Messages for creating DataTable:
- "Blank column name"
- "Duplicated column name"
- "The column index is invalid"
- Error Messages for MessagePackaging Exception
- "Input Message is null"
- "Failed to connect to Server"

- "Failed to disconnect from Server"
- "Failed to open Http Connection"
- "Failed to get Service"
- "Failed to package the message using Serializer and Encoding"

SmartPointer

Spectrum™ Technology Platform provides a class called SmartPointer that uses a simple form of reference counting to help track allocation of dynamic memory and perform memory management task.

For example:

```
SmartPointer<Server> server =new Server();
server.connect();
...
server.disconnect();
```

You do not need to delete the memory for pointer server. SmartPointer handles all memory management for you.

Example Application

The sample code shown below illustrates how to use the C++ ASCII version API.

```
try{

   //Create Server
   SmartPointer<Server> server =new Server();

   //Set server connection properties
   server->setConnectionProperty(Server::HOST, "localhost");
   server->setConnectionProperty(Server::PORT, "10119");
   server->setConnectionProperty(Server::CONNECTION_TYPE , "SOCKET");
   server->setConnectionProperty(Server::ACCOUNT_ID, "guest");
   server->setConnectionProperty(Server::ACCOUNT_PASSWORD, "");

   //Connect to server
   server->connect();

   //Get Service From Server
   SmartPointer<Service> service = server- >getService("ValidateAddress");

   //Create Input Message
   SmartPointer<Message> request = new Message();

   //Fill DataTable in the input message
   SmartPointer<DataTable> dataTable = request->getDataTable();
    SmartPointer<DataRow> row1 = dataTable->newRow();
    row1->set("AddressLine1", "4200 Parliament Place") ;
    row1->set("City", "Lanham");
    row1->set("StateProvince", "Maryland");
    dataTable->addRow(row1);

    SmartPointer<DataRow> row2 = dataTable->newRow();
    row2->set("AddressLine1", "100 Congress");
    row2->set("City", "Austin");
    row2->set("StateProvince", "Texas");
```

```
    dataTable->addRow(row2);

   //Set"option" Properties to the Input Message
    request->putOption("OutputCasing", "M");
    request->putOption("OutputRecordType", "A");

   //Process Input Message, return output Message
    SmartPointer<Message> reply = service->process(request);

    //Disconnect from server
   server->disconnect();

   //Get the result from the resonse message
   SmartPointer<DataTable> returnDataTable = reply->getDataTable();

    vector<string> columnName = returnDataTable- >getColumnNames();
      vector< SmartPointer<DataRow> >::iterator iter =
returnDataTable->iterator();

     for (int i=0; i< returnDataTable->getRowCount(); i++, iter++)
   {
   SmartPointer<DataRow> dataRow = *iter;

   for (int col = 0; col < returnDataTable->getColumnCount(); col++)
   {
   const char* value = dataRow->get(columnName[col].c_str());
   cout << value << "\n";
  }
   }
   }catch(Exception e)
   {
   cout << "Error Occurred, " << e.getErrorMessage();
   }
```

The sample code shown below illustrates how to use the C++ Unicode version API.

```
try{
 //Create Server
 SmartPointer<Server> server =new Server();

 //Set server connection properties
 server->setConnectionProperty(Server::HOST,"localhost");
 server->setConnectionProperty(Server::PORT, "10119");
 server->setConnectionProperty(Server::CONNECTION_TYPE , "SOCKET");
 server->setConnectionProperty(Server::ACCOUNT_ID, "guest");
 server->setConnectionProperty(Server::ACCOUNT_PASSWORD, "");

 //Connect to server
 server->connect();

 //Get Service From Server
 //NOTE: ValidateAddress does not support unicode, but supports
 //characters in Canadian address and International address data files.
 SmartPointer<Service> service = server- >getService("ValidateAddress");

 //Create Input Message
 SmartPointer<Message> request = new Message();

 //Fill DataTable in the input message
 SmartPointer<DataTable> dataTable = request->getDataTable();
```

```
dataTable->addColumn("AddressLine1");
dataTable->addColumn("City");
dataTable->addColumn("PostalCode");
dataTable->addColumn("Country");

SmartPointer<DataRow> row1 = dataTable->newRow();

UnicodeString address1 = "74, Rue Octave Bénard";
row1->set( 0 , address1);
UnicodeString city1 = "Etang-Salé-les-Bains";
row1->set( 1 , city1);
UnicodeString postalCode1 = "97427";
row1->set( 2 , postalCode1);
UnicodeString country1 = "Reunion Island";
row1->set( 3 , country1);

dataTable->addRow(row1);

SmartPointer<DataRow> row2 = dataTable->newRow();
UnicodeString address2 = "Final Av. Panteón Foro Libertador";
row2->set( 0 , address2);
UnicodeString city2 = "Caracas";
row2->set( 1 , city2);
UnicodeString postalCode2 = "1010";
row2->set( 2 , postalCode2);
UnicodeString country2 = "Venezuela";
row2->set( 3 , country2);

dataTable->addRow(row2);

//Set"option" Properties to the Input Message
request->putOption("OutputCasing", "M");
request->putOption("OutputRecordType", "A");

//Process Input Message, return output Message
SmartPointer<Message> reply = service->process(request);

//Disconnect from server
server->disconnect();

//Get the result from the resonse message
SmartPointer<DataTable> returnDataTable = reply->getDataTable();

vector<UnicodeString> columnName = returnDataTable- >getColumnNames();

vector< SmartPointer<DataRow> >::iterator iter = returnDataTable- >iterator();


for (int i=0; i< returnDataTable->getRowCount(); i++, iter++)
{
SmartPointer<DataRow> dataRow = *iter;

for (int col = 0; col < returnDataTable->getColumnCount(); col++)
{
UnicodeString value = dataRow->get(columnName[col]);
wcout <<value.getTerminatedBuffer() <<"\n"; }
}

}catch(Exception e)
{
UnicodeString error = e.getErrorMessage() ;
```

```
wcout << error.getTerminatedBuffer();
}
```

Server

The Server class is used to connect to the server, disconnect from the server, and get the service from the server. The following table summarizes the functions each method performs in the Server class.

Table 27: Server Methods Summary

Method	Function
connect	Connects to the server.
disconnect	Disconnects from the server.
setConnectionProperty	Sets the configuration items for the connection to the server.
getService	Gets the service (i.e., ValidateAddress) from the server.

Note: See the Modules section of this guide for a list of services that may be available to you.

Constructors

Constructors for the Server class are as follows:

- Server()

Destructor

The Destructor for the Server class is:

- ~Server()

Connect

Reads the properties to determine the configuration settings and makes a connection to the server. You can connect via HTTP, HTTPS, or SOCKET.

Note: C++ uses the HTTP, HTTPS, or SOCKET server connection protocol. HTTP and HTTPS logically establish a client connection but do not actually connect to the server until a GetService or

Process method is invoked. The SOCKET protocol establishes a connection to the server when Connect is invoked.

Syntax

```
void connect()
```

Parameters

None.

Results

Establishes client connection to the server.

Example

```
//Create Server
SmartPointer<Server> server =new Server();

//Set server connection properties
server->setConnectionProperty(Server::HOST,"localhost");
server->setConnectionProperty(Server::PORT, "10119");
server->setConnectionProperty(Server::CONNECTION_TYPE , "SOCKET");
server->setConnectionProperty(Server::ACCOUNT_ID, "guest");
server->setConnectionProperty(Server::ACCOUNT_PASSWORD, "");

//Connect to server
server->connect();
```

Disconnect

Disconnects from the server.

Syntax

```
void disconnect()
```

Parameters

None.

Results

Client is disconnected from the server.

Example

```
SmartPointer<Server> server =new Server()
server->connect();
...
server->disconnect();
```

SetConnectionProperty

Establishes the server connection configuration properties, such as host name and length of timeout.

Syntax

ASCII version:

```
void setConnectionProperty(const char* name, const char* value)
```

Unicode version:

```
void setConnectionProperty(const UnicodeString name, const UnicodeString
value)
```

Parameters

- Name — the name of the connection property, such as HOST
- Value — the value for the name of the connection property, such as "www.myhost.com"

Results

The configuration properties for connection to the server are set.

Example

ASCII Version

```
SmartPointer<Server> server =new Server()
server->setConnectionProperty(Server::HOST,"localhost");
server->setConnectionProperty(Server::PORT, "8080");
```

Unicode Version

Same as ASCII, or:

```
SmartPointer<Server> server =new Server()
UnicodeString host="localhost";// Or input unicode string
server->setConnectionProperty(Server::HOST, host);
```

GetService

Gets the service from the server.

Note: See the Component Reference section of this guide for a list of servies that may be available to you.

Syntax

ASCII Version:

```
SmartPointer<Service> getService(const char* serviceName)
```

Unicode Version:

```
SmartPointer<Service> getService(const UnicodeString serviceName)
```

Parameters

- Name of service

Results

Returns the specific service.

Example

ASCII Version

```
// Get Service From Server
SmartPointer<Service> service = server- >getService("ValidateAddress");
```

Unicode Version

Same as ASCII, or:

```
// Get Service From Server
UnicodeString serviceName="ValidateAddress";// Or input unicode string
SmartPointer<Service> service = server->getService(serviceName);
```

Service

The Service class is used to process the message (i.e., send the message to the server and receive a response from the server).

This class has just one method: Process. The Process method is detailed below.

Process

Processes the input message and returns the response message.

Syntax

```
SmartPointer<Message> process(Message* message)
```

Parameters

- Input message

Results

Returns the response message.

Example

```
SmartPointer<Message> reply = service->process(request);
```

Message

The Message class sends your input data and receives your output data from the service. The properties for Message include context entities, such as account ID, account password, service name, and service method; option entities, which are the Service-specific runtime options; and error entities, which are the error class, error message and error stacktrace.

Constructors

Constructors for the Message class are as follows:

- Message()

For example:

```
Message *request = new Message():
```

- Message(const Message&)

For example:

```
Message* request = new Message();
Message anotherMessage = request;
Message message(anotherMessage);
```

Destructor

The Destructor for the Message class is:

- ~Message();

The following table summarizes the functions each method performs in the Message class.

Table 28: Message Methods Summary

Method	Function
getContext	Gets the value of the context entity identified by the name in the context session of the message.
getContext	Gets the Map that contains all of the context entries.
putContext	Sets the value of the context entity identified by the name in the context session of the message. If there is an existing value present for the entity identified by the name, it is replaced.

Method	Function
putContext	Adds the new context properties to the current context properties.
setContext	Overwrites the current context properties with the new context properties.
getOption	Gets the value of the option entity identified by the name in the option session of the message.
getOptions	Gets the Map that contains all of the option entries.
putOption	Sets the value of the option entity identified by the name in the option session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putOptions	Adds the new option properties to the current option properties.
setOptions	Overwrites the current option properties with the new option properties.
getError	Gets the error message.
getDataTable	Gets the DataTable from the message.

GetContext

Gets the value of the context entity identified by the name in the context session of the message.

Syntax

ASCII Version

```
const char* getContext(const char* name)
```

Unicode Version

```
const UnicodeString getContext(const UnicodeString name)
```

Parameters

- The name whose associated value is to be returned

Results

Returns the value for the name in the context entity. If the name does not exist, the method returns empty string.

Example

ASCII Version

```
const char* value= msg->getContext(Server::ACCOUNT_ID);
```

Unicode Version

Same as ASCII or:

```
UnicodeString name= Server::ACCOUNT_ID;// Or input unicode string
UnicodeString value= msg->getContext(name);
```

GetContext

Gets the Map that contains all of the context entries.

Syntax

ASCII Version

```
map<string , string> getContext()
```

Unicode Version

```
map< UnicodeString, UnicodeString > getContext()
```

Parameters

None.

Results

Returns the map that contains all of the context entries.

Example

ASCII Version

```
map<string , string> context = message->getContext();
```

Unicode Version

```
map< UnicodeString, UnicodeString > context = message- >getContext();
```

PutContext

Sets the value for given name in the context properties. If there is an existing value present for the entity identified by the name, it is replaced. Context properties include the following constants: account ID, account password, service name, service key, and request ID.

Syntax

ASCII Version

```
void putContext(const char* name, const char* value)
```

Unicode Version

```
void putContext(const UnicodeString name, const UnicodeString value)
```

Parameters

- Name with which the specified value is to be associated.
- Value to be associated with the specified name

Example

ASCII Version

```
message->putContext(Message.CONTEXT_ACCOUNT_ID, "user1");
```

Unicode Version

Same as ASCII or:

```
UnicodeString account="user1" ;// Or input unicode string
message->putContext(Message.CONTEXT_ACCOUNT_ID, account);
```

PutContext

Adds the new context properties to the current context properties.

Syntax

ASCII Version

```
void putContext(map<string , string> context)
```

Unicode Version

```
void putContext(map< UnicodeString, UnicodeString > context)
```

Parameters

- The new context map to be added to the current context map

Example

ASCII Version

```
map<string , string> context ;
//more code
message->putContext(context);
```

Unicode Version

```
map< UnicodeString, UnicodeString > context ;
//more code
message->putContext(context);
```

SetContext

Overwrites the current context properties with the new context properties.

Syntax

ASCII Version

```
void setContext(map<string , string> context)
```

Unicode Version

```
void setContext(map< UnicodeString, UnicodeString > context)
```

Parameters

- The new context map to be used to replace the current context map

Example

ASCII Version

```
map<string , string> context ;
//more code
message->setContext(context);
```

Unicode Version

```
map< UnicodeString, UnicodeString > context ;
//more code
message->setContext(context);
```

GetOption

Gets the value of the option entity identified by name in the option section of the message. Option entities include the service-specific runtime options, such as output casing, output data format, and so on.

Note: See **Modules** on page 241 for a complete list of options for services that may be available to you.

Syntax

ASCII Version

```
const char* getOption(const char* name)
```

Unicode Version

```
const UnicodeString getOption(const UnicodeString name)
```

Parameters

- The name whose associated value is to be returned

Results

Returns the value for the name in the context entity. If the name does not exist, the method returns empty string.

Example

ASCII Version

```
const char* value = message->getOption("OutputCasing");
```

Unicode Version

Same as ASCII or:

```
UnicodeString option="OutputCasing"; // Or input unicode string
UnicodeString value= message->getOption(option);
```

GetOptions

Gets the map that contains all of the option entries.

Syntax

ASCII Version

```
map<string , string> getOptions()
```

Unicode Version

```
map< UnicodeString, UnicodeString > getOptions()
```

Parameters

None.

Results

Returns the map that contains all of the option entries.

Example

ASCII Version

```
const char* value = message->getOption("OutputCasing");
```

Unicode Version

```
UnicodeString option="OutputCasing"; //or input Unicode string
UnicodeString value= message->getOption(option);
```

PutOption

Sets the value for given name in the option properties. If there is an existing value present for the entity identified by the name, it is replaced. Option properties are the service-specific run-time options.

Note: See **Modules** on page 241 for a complete list of options for each service.

Syntax

ASCII Version

```
void putOption(const char* name, const char* value)
```

Unicode Version

```
void putOption(const UnicodeString name, const UnicodeString value)
```

Parameters

- Name with which the specified value is to be associated
- Value to be associated with the specified name

Example

ASCII Version

```
message->putOption("OutputCasing", "M");
```

Unicode Version

Same as ASCII or:

```
UnicodeString option="M"; // Or input unicode string
message->putOption("OutputCasing", option);
```

PutOptions

Adds the new option properties to the current option properties.

Syntax

ASCII Version

```
void putOptions(map<string , string> options)
```

Unicode Version

```
void putOptions(map< UnicodeString, UnicodeString > options)
```

Parameters

- The new option map to be added to the current option properties

Example

ASCII Version

```
map<string , string> options ;
//more code
message->putOptions(options);
```

Unicode Version

```
map< UnicodeString, UnicodeString > options ;
//more code
message->putOptions(options);
```

SetOptions

Overwrites the current option properties with the new option properties.

Syntax

ASCII Version

```
void setOptions(map<string , string> options)
```

Unicode Version

```
void setOptions(map< UnicodeString, UnicodeString > options)
```

Parameters

- The new option map to be used to replace the current option map

Example

ASCII Version

```
map<string , string> options ;
//more code
message->setOptions(options);
```

Unicode Version

```
map< UnicodeString, UnicodeString > options ;
//more code
message->setOptions(options);
```

GetError

Gets the error message from the message.

Syntax

ASCII Version

```
string getError()
```

Unicode Version

```
UnicodeString getError()
```

Parameters

None.

Results

Returns the error message in message

Example

ASCII Version

```
String error = message->getError();
```

Unicode Version

```
UnicodeString error = message->getError();
```

GetDataTable

Gets the DataTable in the message.

Syntax

```
SmartPointer<DataTable> getDataTable()
```

Parameters

None.

Example

```
SmartPointer<DataTable> dataTable
= message->getDataTable();
```

DataTable

DataTable contains the records for the input and output data.

Constructors

Constructors for the DataTable class are as follows:

- DataTable()

For example:

```
DataTable* dataTable = new DataTable()
```

Destructor

The Destructor for the DataTable class is:

- ~DataTable();

The following table summarizes the functions each method performs in the DataTable class.

Table 29: DataTable Methods Summary

Method	Function
addColumn	Adds the new column.
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
clear	Clears the data in DataTable.
iterator	An iterator that contains all DataRows in the DataTable.
addRow	Adds a DataRow to the DataTable.
newRow	Creates a new DataRow in the DataTable.
getRowCount	Gets the number of the DataRows in this DataTable.
merge	Merges the given DataTable and the current DataTable.

AddColumn

Adds the new column.

Note: See **Modules** on page 241 for a complete list of valid columnName values for each service.

Syntax

ASCII Version

```
int addColumn(const char* columnName)
```

Unicode Version

```
int addColumn(const UnicodeString columnName)
```

Parameters

- Column name

Results

- Returns the index of column

Exceptions

- Blank column name
- Duplicate column name

Example

ASCII Version

```
SmartPointer<DataTable> dataTable = message.getDataTable();
dataTable->addColumn("Address");
dataTable->addColumn("City");
```

Unicode Version

Same as ASCII or:

```
SmartPointer<DataTable> dataTable = message.getDataTable();
UnicodeString columnName="Address"; // Or input unicode string
dataTable->addColumn(columnName);
```

GetColumnNames

Gets all the column names.

Syntax

ASCII Version

```
vector<string> getColumnNames();
```

Unicode Version

```
vector<UnicodeString> getColumnNames();
```

Parameters

None.

Results

Returns the vector of column names

Example

ASCII Version

```
vector<string> columnNames = dataTable->getColumnNames();
```

Unicode Version

```
vector<UnicodeString> columnNames = dataTable->getColumnNames();
```

GetColumnIndex

Gets the corresponding column index.

Note: See **Modules** on page 241 for a complete list of valid columnName values for each service.

Syntax

ASCII Version

```
int getColumnIndex(const char* columnName)
```

Unicode Version

```
int getColumnIndex(const UnicodeString columnName)
```

Parameter

- Column name

Results

Returns the corresponding column index.

Example

ASCII Version

```
int columnIndex = dataTable->getColumnIndex ("City");
```

Unicode Version

Same as ASCII or:

```
UnicodeString columnName="City"; // Or input unicode string
int columnIndex = dataTable->getColumnIndex (columnName);
```

GetColumnCount

Gets the number of columns.

Syntax

```
int getColumnCount()
```

Parameter

None.

Results

Returns the number of columns.

Example

```
int columnCount = dataTable->getColumnCount ();
```

Clear

Clears the data in DataTable.

Syntax

```
void clear()
```

Parameters

None.

Example

```
dataTable->clear();
```

Iterator

An iterator that contains all DataRows in the DataTable.

Syntax

```
vector< SmartPointer<DataRow> >::iterator iterator()
```

Parameters

None.

Results

Returns an iterator that contains all DataRows in the DataTable.

Example

```
vector<string> columnName
= returnDataTable->getColumnNames();

vector< SmartPointer<DataRow> >::iterator theIterator
= returnDataTable->iterator();

for (int i=0; i< returnDataTable->getRowCount();
i++, theIterator++)
{
SmartPointer<DataRow> dataRow = *theIterator;

for (int col = 0;
col < returnDataTable->getColumnCount(); col++)
 {
 const char* value = dataRow->get(columnName[col].c_str());
 }
 }
```

AddRow

Adds a DataRow to the DataTable.

Syntax

```
void addRow( SmartPointer<DataRow> dataRow)
```

Parameters

- DataRow to be added to the DataTable

Example

```
SmartPointer<DataRow> newRow = dataTable->newRow();
 newRow->set( 0 , "10535 Boyer");
 newRow->set( 1 , "Austin");
 newRow->set( 2 , "Texas");
 dataTable->addRow(newRow);
```

NewRow

Creates a new DataRow in the DataTable.

Syntax

```
SmartPointer<DataRow> newRow()
```

Results

Returns the new created DataRow

Example

```
SmartPointer<DataRow> newRow = dataTable->newRow();
 newRow->set( 0 , "10535 Boyer");
 newRow->set( 1 , "Austin");
 newRow->set( 2 , "Texas");
 dataTable->addRow(newRow);
```

GetRowCount

Gets the number of the DataRows in this DataTable.

Syntax

```
int getRowCount()
```

Results

Returns the number of the DataRows in this DataTable.

Example

```
int rowCount = dataTable->getRowCount();
```

Merge

Merges the given DataTable and the current DataTable.

Syntax

```
void merge(DataTable* other)
```

Parameters

- Other DataTable to be merged with the current DataTable

Example

```
DataTable* otherDataTable = new DataTable();
dataTable->merge(otherDataTable);
```

DataRow

DataRow contains the record for the input and output data.

Constructor

Constructors for the DataRow class are as follows:

- DataRow ()

For example:

```
DataRow * dataRow = new DataRow();
```

- DataRow(const DataRow&)

For example:

```
DataRow* dataRow = new DataRow();
DataRow anotheDataRow = dataRow;
DataRow newDataRow(anotheDataRow);
```

Destructor

The Destructor for the DataRow class is:

- ~ DataRow();

The following table summarizes the functions each method performs in the DataRow class.

Table 30: DataRow Methods Summary

Method	Function
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
get	Gets the value from the fields array by the column index in this DataRow.
get	Gets the value from the fields array by the column name in this DataRow.
merge	Merges the given DataTable and the current DataTable.
set	Sets the value for the corresponding column name for the DataRow. If the value for the name exists, the old value is replaced.
set	Sets the value for the corresponding column index for the DataRow. If the value for the name exists, the old value is replaced.
addChild	Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow Collection, otherwise a new Collection will be created with the supplied DataRow as its only element.
getChildren	Retrieves the child rows from a named relationship.
listChildNames	Retrieves all of the names of the named parent/child relationships.
setChildren	Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

GetColumnNames

Gets all the column names.

Syntax

ASCII Version

```
vector<string> getColumnNames()
```

Unicode Version

```
vector<UnicodeString> getColumnNames()
```

Parameters

None.

Results

Returns the vector of column names

Example

ASCII Version

```
vector<string> columnNames = dataRow->getColumnNames();
```

Unicode Version

```
vector<UnicodeString> columnNames = dataRow->getColumnNames();
```

GetColumnIndex

Gets the corresponding column index.

Syntax

ASCII Version

```
int getColumnIndex(const char* columnName)
```

Unicode Version

```
int getColumnIndex(const UnicodeString columnName)
```

Parameter

- Column name

Results

Returns the corresponding column index.

Example

ASCII Version

```
int columnIndex = dataRow->getColumnIndex ("City");
```

Unicode Version

Same as ASCII or:

```
UnicodeString columnName="City"; // Or input unicode string
int columnIndex = dataRow->getColumnIndex (columnName);
```

GetColumnCount

Gets the number of columns.

Syntax

```
int getColumnCount()
```

Parameter

None.

Results

Returns the number of columns.

Example

```
int columnCount = dataRow->getColumnCount ();
```

Get

Gets the value from the fields array by the column index in this DataRow.

Note: See **Modules** on page 241 for a complete list of valid columnName values for each service.

Syntax

ASCII Version

```
const char* get(int index)
```

Unicode Version

```
const UnicodeString get(int index)
```

Parameters

- Index with which the specified value is to be associated

Results

Returns the value for the column index in the DataRow, returns empty string if the index in invalid.

Example

ASCII Version

```
const char* value = dataRow->get(1);
```

Unicode Version

```
const UnicodeString value = dataRow->get(1);
```

Get

Gets the value from the fields array by the column name in this DataRow

Syntax

ASCII Version

```
const char* get(const char* columnName)
```

Unicode Version

```
const UnicodeString get(const UnicodeString columnName)
```

Parameters

- Name with which the specified value is to be associated

Results

Returns the value for the column name in the DataRow, returns empty string if the column name does not exist.

Example

ASCII Version

```
const char* value = dataRow->get("City");
```

Unicode Version

Same as ASCII, or:

```
UnicodeString columnName="City"; // Or input unicode string
const UnicodeString value = dataRow->get(columnName);
```

Merge

Merges the given DataRow and the current DataRow.

Syntax

```
void merge(DataRow* other)
```

Parameters

- Other DataRow to be merged with the current DataRow

Example

```
DataRow* otherDataRow = new DataRow();
DataRow->merge(otherDataRow);
```

Set

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Note: See **Modules** on page 241 for a complete list of valid column name values for each service.

Syntax

ASCII Version

```
void set(const char* columnName, const char* value)
```

Unicode Version

```
void set(const UnicodeString columnName, const UnicodeString value)
```

Parameters

- Name with which the specified value is to be associated
- Value to be associated with the specified name

Exceptions

- Blank column name
- Duplicate column name

Example

ASCII Version

```
SmartPointer<DataRow> newRow = dataTable->newRow();
newRow->set( "AddressLine1" , "10535 Boyer");
newRow->set( "City" , "Austin");
newRow->set( "State" , "Texas");
```

Unicode Version

Same as ASCII or:

```
SmartPointer<DataRow> newRow = dataTable->newRow();
UnicodeString address="10535 Boyer"; // Or input unicode string
newRow->set( "AddressLine1" , address);
```

Set

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

ASCII Version

```
void set(int index, const char* value)
```

Unicode Version

```
void set(int index, const UnicodeString value)
```

Parameters

- Column index with which the specified value is to be associated
- Value to be associated with the specified name

Exceptions

- The column index is invalid.

Example

ASCII Version

```
SmartPointer<DataRow> newRow = dataTable->newRow();
newRow->set( 0 , "10535 Boyer");
newRow->set( 1 , "Austin");
newRow->set( 2 , "Texas");
```

Unicode Version

Same as ASCII or:

```
SmartPointer<DataRow> newRow = dataTable->newRow();
UnicodeString address="10535 Boyer"; // Or input unicode string
newRow->set( 0 , address);
```

AddChild

Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow collection. Otherwise, a new collection will be created with the supplied DataRow as its only element.

Syntax

ASCII Version

```
void addChild(const char* childName, SmartPointer<DataRow> childDataRow)
```

Unicode Version

```
void addChild(const UnicodeString childName, SmartPointer<DataRow>
childDataRow)
```

Parameters

- The name of the parent/child relationship (for example, "Flood Plain Data," "References, " "Used By")
- The DataRow to be added to the relationship.

Example

```
SmartPointer<DataRow> childDataRow =new DataRow();
childDataRow ->set("Address", "100 Congress");
childDataRow ->set("City", "Austin");
SmartPointer<DataRow> dataRow =new DataRow();
dataRow->addChild("child1", childDataRow );
```

GetChildren

Retrieves the child rows from a named relationship.

Syntax

ASCII Version

```
list< SmartPointer<DataRow> > getChildren(const char* childName)
```

Unicode Version

```
list< SmartPointer<DataRow> > getChildren(const UnicodeString childName)
```

Parameters

- The name of the parent/child relationship, e.g. "Flood Plain Data", "References", "Used By", etc.

Results

Returns the child rows from the named relationship.

Example

```
list< SmartPointer<DataRow> > rowsChild2= dataRow- >getChildren("child2");
```

ListChildNames

Retrieves all of the names of the named parent/child relationships.

Syntax

ASCII Version

```
list<string> listChildNames()
```

Unicode Version

```
list<UnicodeString> listChildNames()
```

Results

Returns the set of the names of the named parent/child relationships.

Example

```
list<G1CLIENT_STRING> names = dataRow->listChildNames();
```

SetChildren

Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

Syntax

ASCII Version

```
list< SmartPointer<DataRow> > setChildren(const char* childName, list<
SmartPointer<DataRow> > dataRows)
```

Unicode Version

```
list< SmartPointer<DataRow> > setChildren(const UnicodeString childName,
list< SmartPointer<DataRow> > dataRows)
```

Results

Returns the set of the names of the named parent/child relationships.

Example

```
SmartPointer<DataRow> dataRow1=new DataRow();
dataRow1->set("Address", "100 Congress");
dataRow1->set("City", "Austin");
SmartPointer<DataRow> dataRow2=new DataRow();
dataRow2->set("Address", "200 Congress");
dataRow2->set("City", "Austin");
list< SmartPointer<DataRow> > rows ;
rows.push_back(dataRow1);
rows.push_back(dataRow2);
list< SmartPointer<DataRow> > rowsNewChildren = dataRowSpt-
>setChildren("child1", rows);
```

6

The COM API

In this section:

- Introduction 150
- Server 153
- Service 156
- Message 157
- DataTable 164
- DataRow 170
- Map 177

Introduction

The term Component Object Model (COM) refers to an open architecture for cross-platform development of client/server applications based on object-oriented technology. COM is a way of creating reusable software components. Clients have access to an object through interfaces implemented on the object. In other words, objects are the communication medium between client and server. The Component Object Model provides a flexible way for building distributed object-oriented systems. COM objects are language-independent, can be shipped in binary form, can be upgraded without requiring changes to the existing integrated code, and can be transparently relocated on a network. Because of these qualities, COM objects are extremely flexible and can be adapted to add specific functionality to almost any Windows-based client-server system.

Note: The examples shown in this chapter are written in Visual Basic.

The Spectrum™ Technology Platform COM API consists of the following interfaces:

- Server
- Service
- Message
- DataTable
- DataRow
- Map

These interfaces are described in detail in the following sections of this chapter.

Constants

The COM API uses two sets of constants. The first set is for the Server object, described in the table below.

Table 31: Constants for the Server Component

Constant Name	Description/Default	Example
SERVER.HOST	String for server host name. Default is "localhost".	65.89.200.89
SERVER.PORT	String for server port. Default is "8080".	10119
SERVER.ACCOUNT_ID	String for server account ID. No default value.	user1
SERVER.ACCOUNT_PASSWORD	String for server account password. No default value.	user1

Constant Name	Description/Default	Example
SERVER.CONNECTION_TIMEOUT	String for server connection timeout, in milliseconds. Default is "5000".	50000
SERVER.CONNECTION_TYPE	String for server connection type. Currently only supports HTTP, HTTPS, or SOCKET. Default is "HTTP".	HTTP(S)
SERVER.PROXY_HOST	String for proxy server host name. No default value.	192.168.1.77
SERVER.PROXY_PORT	String for proxy server port. No default value.	8080
SERVER.PROXY_USER	String for proxy server account ID. No default value.	user1
SERVER.PROXY_PASSWORD	String for proxy server account password. No default value.	user1

The second set of constants is for the Message component.

Table 32: Constants for the Message Component

Constant Name	Description	Example
MESSAGE.CONTEXT_ACCOUNT_ID	String for message context account ID.	user1
MESSAGE.CONTEXT_ACCOUNT_PASSWORD	String for message context account password.	user1
MESSAGE.CONTEXT_SERVICE_NAME	String for message context service name.	echoservice

Error Messages

The COM API uses the following error messages:

- Error Messages for Connection:
- "Connection type not supported"
- "Client timeout"
- Error Messages for creating DataTable:

- "Blank column name"
- "Duplicated column name"
- "The column index is invalid"
- Error Messages for Message Packaging Exception
- "Input Message is null"
- "Failed to connect to Server"
- "Failed to disconnect to Server"
- "Failed to open Http Connection"
- "Failed to get Service"
- "Failed to package the message using Serializer and Encoding"

For example:

```
On Error GoTo ErrorHandler
  Dim server As New G1CLIENTLib.server
  server.setConnectionProperty server.HOST, "localhost"
  server.setConnectionProperty server.Port, "8080"
  'Making connection to the server
  server.Connect
  ...
  Exit Sub
  ErrorHandler:
  MsgBox Err.Description
```

Example Application

The sample code shown below illustrates how to use the COM API.

```
On Error GoTo ErrorHandler

 Dim server As New G1CLIENTLib.server
 Dim service As G1CLIENTLib.service
 Dim requestMsg As New G1CLIENTLib.Message
 Dim replyMsg As G1CLIENTLib.Message
 Dim dataTable As G1CLIENTLib.dataTable
 Dim newRow As G1CLIENTLib.dataRow
 Dim returnDataTable As G1CLIENTLib.dataTable
 Dim row As G1CLIENTLib.DataRow
 Dim sColumnNames() As String
 Dim sColumnName As String
 Dim sFieldValue As String
 Dim rows() As Variant
 Dim nRow As Integer
 Dim nColumn As Integer
 'Set server connection properties
 server.setConnectionProperty server.HOST, "localhost"
 server.setConnectionProperty server.Port, "10119"
 server.setConnectionProperty server.CONNECTION_TYPE, "SOCKET"
 server.setConnectionProperty server.ACCOUNT_ID, "guest"
 server.setConnectionProperty server.ACCOUNT_PASSWORD, ""

 'Connect to server
 server.Connect

 'Get the service from the server
 Set service = server.getService("ValidateAddress")
```

```
 'Fill DataTable in the input message
 Set dataTable = requestMsg.getDataTable
 dataTable.addColumn ("AddressLine1")
 dataTable.addColumn ("City")
 dataTable.addColumn ("StateProvince")

 Set newRow = dataTable.newRow
 newRow.setByIndex 0, "10535 Boyer"
 newRow.setByIndex 1, "Austin"
 newRow.setByIndex 2, "Texas"
 dataTable.addRow newRow

 'Set"option" Properties to the Input Message
 requestMsg.putOption "OutputCasing", "M"
 requestMsg.putOption "OutputRecordType", "A"

 'Process Input Message, return output Message
 Set replyMsg = service.process(requestMsg)

 'Disconnect from the server
 server.disconnect

 'Get the result from the resonse message
 Set returnDataTable = replyMsg.getDataTable
 ReDim rows(returnDataTable.getRowCount) As Variant

 rows = returnDataTable.iterator

 ReDim sColumnNames(returnDataTable.getColumnCount) As String
 sColumnNames = returnDataTable.getColumnNames

 For nRow = 0 To returnDataTable.getRowCount - 1
Set row = rows(nRow)

For nColumn = 0 To row.getColumnCount - 1
  sColumnName = sColumnNames(nColumn)
  sFieldValue = row.getByName(sColumnName)
Next

 Next

  Exit Sub

  ErrorHandler:

MsgBox Err.Description
```

Server

Use the Server object to connect to the server, disconnect from the server, and get the service from the server. The following table summarizes the functions each method performs in the Server object.

Table 33: Server Methods Summary

Method	Function
connect	Connects to the server.
disconnect	Disconnects from the server.
getService	Gets the service (i.e., ValidateAddress) from the server.
setConnectionProperty	Sets the configuration items for the connection to the server.

Note: See the Modules section of this guide for a list of services that may be available to you.

Connect

Connects to the server. You can connect via HTTP or SOCKET.

Note: COM uses the HTTP, HTTPS, or SOCKET server connection protocol. HTTP and HTTPS logically establish a client connection but do not actually connect to the server until a GetService or Process method is invoked. The SOCKET protocol establishes a connection to the server when Connect is invoked.

Syntax

```
Sub connect()
```

Parameters

None.

Results

None.

Exception

Connection type not supported.

Example

```
Dim server As New G1CLIENTLib.server
server.connect
```

Disconnect

Disconnects from the server.

Syntax

```
Sub disconnect()
```

Parameters

None.

Results

None.

Examples

```
Dim server As New G1CLIENTLib.server
server.disconnect
```

GetService

Gets the service (such as ValidateAddress) from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Syntax

```
Function getService(serviceName As String) As Service
```

Parameters

- serviceName - the name of the service which the client requires

Results

The requested service or NULL if the service does not exist.

Exceptions

- ERROR_FAIL_TO_GET_SERVICE — if there is no connection to the server.

Example

```
Dim server As New G1CLIENTLib.server
Dim service As G1CLIENTLib.service
...
'get the service from the server
Set service = server.getService("ValidateAddress")
```

SetConnectionProperty

Establishes the server connection configuration properties, such as host name and length of timeout.

Syntax

```
Sub setConnectionProperty(name As String, value As String)
```

Parameters

- Name — the name of the connection property, such as HOST
- Value — the value for the name of the connection property, such as "www.myhost.com"

Results

Return codes — none.

Exceptions

- ERROR-INVALID-COLUMN_NAME — an empty or null column name.
- ERROR_INVALID_VALUE — A null value.

Example

```
set connection properties
Dim server As New G1CLIENTLib.server

server.setConnectionProperty server.HOST,"localhost"
server.setConnectionProperty server.PORT, "8080"
```

Service

Service calls the service and processes the message you are sending (in other words, it sends the input message and receives the response).

There is only one method in the Service object.

Table 34: Service Methods Summary

Method	Function
process	Processes the input message and gets back the response message from the server.

Process

Processes the input message and gets back the response message from the server.

Syntax

```
Function process(IRequest As Message) As Message
```

Parameters

- iRequest— the input message object that contains the "option" setting and the DataTable

Results

Returns the response message for the request.

Exceptions:

- ERROR_NULL_INPUT_MESSAGE — Request message is null.

Example

```
Dim service As New G1CLIENTLib.service
Dim replyMsg As G1CLIENTLib.Message
...
'Process the message and return back the response message
Set replyMsg = service.process(requestMsg)
```

Message

Messages are used to send your input data and receive your output data from the service. The properties for Message include "context entities, such as account ID, account password, service name, and service method; "option entities, which are the service-specific runtime options; and "error entities, which are the error class, error message and error stacktrace.

The following table summarizes the functions each method performs in the Message object.

Table 35: Message Methods Summary

Method	Function
getContext	Gets the value of the context entity identified by name in the context section of the message.
getContextMap	Gets the map that contains all of the context entries.

Method	Function
putContext	Sets the value of the context entity identified by name in the context session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putContextMap	Add the new context properties to the current context properties.
setContextMap	Overwrite the current context properties with the new context properties.
getOption	Gets the value of the option entity identified by name in the option section of the message.
getOptions	Gets the map that contains all of the option entries.
putOption	Sets the value of the option entity identified by name in the option section of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putOptions	Add the new option properties to the current option properties.
setOptions	Overwrite the current option properties with the new option properties.
getError	Gets the error from the error message.
getDataTable	Gets the specified data table from the message.

GetContext

Gets the value of the context entity identified by name in the context section of the message. "Context" entities include the following constants: account ID, account password, service name, and service method.

Syntax

```
Function getContext(name As String) As String
```

Parameters

- Name—the name whose associated value is to be returned

Results

String — the value of the named entity or empty string if the named entity does not exist.

Example

```
Dim msg As New G1CLIENTLib.Message
Dim accountID As String

accountID = msg.getContext(msg.CONTEXT_ACCOUNT_ID)
```

GetContextMap

Gets the Map that contains all of the context entries.

Syntax

```
Function getContextMap() As Map
```

Parameters

- None

Results

Returns the Map that contains all of the context entries.

Example

```
Dim map As G1CLIENTLib.Map
Dim requestMsg As New G1CLIENTLib.Message
Dim sKey As String
Dim sValue As String

requestMsg.putContext
 requestMsg.CONTEXT_ACCOUNT_ID, "admin"
requestMsg.putContext
 requestMsg.CONTEXT_ACCOUNT_PASSWORD, "admin"

Set map = requestMsg.getContextMap

map.Reset
While (map.Next)
 sKey = map.getKey
 sValue = map.getValue
Wend
```

PutContext

Sets the value for the given name in the context properties. If there is an existing value present for the entity identified by the name, it is replaced. "Context" properties include the following constants: account ID, account password, service name, and service method.

Syntax

```
Sub putContext(name As String, value As String)
```

Parameters

- Name—the name with which the specified value is to be associated
- Value—value to be associated with the specific name.

Results

None.

Example

```
Dim requestMsg As New G1CLIENTLib.Message

requestMsg.putContext
 requestMsg.CONTEXT_ACOUNT_ID, "admin"
requestMsg.putContext
 requestMsg.CONTEXT_ACOUNT_PASSWORD, "admin"
```

PutContextMap

Adds the new context properties to the current context properties.

Syntax

```
Sub putContextMap(context As Map)
```

Parameters

- The new context map to be added to the current context map

Results

None.

Example

```
Dim map As New G1CLIENTLib.Map
Dim requestMsg As New G1UBCAPICOMLib.Message

map.Insert requestMsg.CONTEXT_ACCOUNT_ID, "admin"
map.Insert requestMsg.CONTEXT_ACOUNT_PASSWORD, "admin"

requestMsg.putContextMap map
```

SetContextMap

Overwrites the current context properties with the new context properties.

Syntax

```
Sub setContextMap(context As Map)
```

Parameters

- The new context map to replace the current context map

Results

None.

Example

```
Dim map As New G1CLIENTLib.Map
Dim requestMsg As New G1UBCAPICOMLib.Message

map.Insert requestMsg.CONTEXT_ACCOUNT_ID, "admin"
map.Insert requestMsg.CONTEXT_ACOUNT_PASSWORD, "admin"

requestMsg.setContextMap map
```

GetOption

Gets the value of the option entity identified by name in the option section of the message. Option entities include the service-specific runtime options, such as output casing, output data format, and so on.

Syntax

```
Function getOption(name As String) As String
```

Parameters

- Name — the name whose associated value is to be returned

Results

- String — the value of the named entity or empty string if the named entity does not exist.

Example

```
Dim msg As New G1CLIENTLib.Message
Dim optionValue As String

OptionValue = msg.getOption("OutputCasing")
```

GetOptions

Gets the map that contains all of the option entries.

Syntax

```
Function getOptions() As Map
```

Parameters

- None

Results

Returns the map that contains all of the option entries.

Example

```
Dim map As New G1CLIENTLib.Map
Dim requestMsg As New G1CLIENTLib.Message
Dim sKey As String
Dim sValue As String

requestMsg.putOption "OutputCasing", "M"
requestMsg.putOption "OutputRecordType", "A"

Set map = requestMsg.getOptions

map.Reset
While (map.Next)
 sKey = map.getKey
 sValue = map.getValue
Wend
```

PutOption

Sets the value for the given name in the option properties. If there is an existing value present for the entity identified by the name, it is replaced. "Option" properties are the service-specific runtime options.

Syntax

```
Sub putOption(name As String, value As String)
```

Parameters

- Name—the name with which the specified value is to be associated
- Value—value to be associated with the specific name.

Example

```
Dim requestMsg As New G1CLIENTLib.Message

requestMsg.putOption "OutputCasing", "M"
requestMsg.putOption "OutputRecordType", "A"
```

PutOptions

Adds the new option properties to the current option properties.

Syntax

```
Sub putOptions(options As Map)
```

Parameters

- The new option map to be added to the current option properties

Example

```
Dim map As New G1CLIENTLib.Map
Dim requestMsg As New G1CLIENTLib.Message

map.Insert "OutputCasing", "M"
map.Insert "OutputRecordType", "A"

requestMsg.putOptions map
```

SetOptions

Overwrites the current option properties with the new option properties.

Syntax

```
Sub setOptions(options As Map)
```

Parameters

- The new option map to replace the current option map

Example

```
Dim map As New G1CLIENTLib.Map
Dim requestMsg As New G1CLIENTLib.Message

map.Insert "OutputCasing", "M"
map.Insert "OutputRecordType", "A"

requestMsg.setOptions map
```

GetError

Gets the error from the error message.

Syntax

```
Function getError() As String
```

Parameters

- None

Results

Returns the error message in the message.

Example

```
Dim sErrorMessage As String
...

sErrorMessage = replyMsg.getError()
```

GetDataTable

Gets the DataTable in the message.

Syntax

```
Function getDataTable() As DataTable
```

Parameters

- None.

Example

```
Dim DataTable AS G1CLIENTLib.dataTable

Set DataTable = message.getDataTable
```

DataTable

DataTable contains the records for your input and output data. Using the methods associated with this object, you define the column names for your output and add rows to the DataTable.

The following table summarizes the functions each method performs in the DataTable object.

Table 36: DataTable Methods Summary

Method	Function
addColumn	Adds a new column to the DataTable.
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
clear	Clears all the data in the DataTable.

Method	Function
iterator	An iterator that contains all the rows in the DataTable.
addRow	Adds a DataRow to the DataTable.
newRow	Creates a new DataRow in the DataTable.
getRowCount	Gets the number of DataRows in the DataTable.
merge	Merge the given DataTable and the current DataTable.

These methods are discussed in more detail in the following sections.

AddColumn

Adds the new column to the DataTable.

Syntax

```
Function addColumn(columnName As String) As Integer
```

Parameters

- Column name

Results

Returns the index of the column

Exceptions

- Blank column name
- Duplicate column name

Example

```
Dim dataTable As G1CLIENTLib.dataTable

dataTable.addcolumn "AddressLine1"
dataTable.addcolumn "City"
```

GetColumnNames

Gets all the column names.

Syntax

```
Syntax Function getColumnNames() As String()
```

Parameters

- None

Results

Returns the array of column names.

Example

```
Dim sColumnNames() As String
Dim sColumnName As String
Dim nColumn As Integer

ReDim sColumnNames(returnDataTable.getColumnCount) As String
sColumnNames = returnDataTable.getColumnNames

For nColumn = 0 To dataRow.getColumnCount - 1
sColumnName = sColumnNames (nColumn)
Next
```

GetColumnIndex

Gets the corresponding column index.

Syntax

```
Function getColumnIndex(columnName As String) As Integer
```

Parameters

- Column name

Results

Returns the corresponding column index.

Example

```
Dim nIndex As Integer
nIndex = dataTable.getColumnIndex("AddressLine1")
```

GetColumnCount

Gets the number of columns in the DataTable.

Syntax

```
Function getColumnCount() As Integer
```

Parameters

- None

Results

Returns the number of columns.

Example

```
Dim nColumnCount As Integer
nColumnCount = dataTable.getColumnCount()
```

Clear

Clears the data in the DataTable.

Syntax

```
Sub clear()
```

Parameters

- None

Example

```
dataTable.clear()
```

Iterator

An iterator that contains all DataRows in the DataTable.

Syntax

```
Syntax Function iterator() As DataRow()
```

Parameters

- None

Results

Returns an iterator that contains all DataRows in the DataTable.

Example

```
Dim returnDataTable As G1CLIENTLib.dataTable
  Dim row As G1CLIENTLib.DataRow
  Dim sColumnName As String
  Dim sFieldValue As String
  Dim rows() As Variant
  Dim nRow As Integer
```

```
  Dim nColumn As Integer

  'Get the result from the resonse message
  Set returnDataTable = replyMsg.getDataTable
  ReDim rows(returnDataTable.getRowCount) As Variant

  rows = returnDataTable.iterator

  For nRow = 0 To returnDataTable.getRowCount - 1
 Set row = rows(nRow)

 For nColumn = 0 To row.getColumnCount - 1
sColumnName = row.getColumnNames(nColumn)
sFieldValue = row.getByName(sColumnName)
 Next

 Next
```

AddRow

Adds a DataRow to the DataTable.

Syntax

```
Sub addRow(DataRow As DataRow)
```

Parameters

- DataRow to be added to the DataTable

Results

None.

Example

```
Dim dataTable As G1CLIENTLib.dataTable
Dim newRow As G1CLIENTlib.DataRow

Set dataTable=requestMsg.getDataTable
dataTable.addColumn("AddressLine1")
dataTable.addColumn("City")
dataTable.addColumn("State")
Set newRow=dataTable.newRow
newRow.setByIndex 0, "10535 Boyer"
newRow.setByIndex 1, "Austin"
newRow.setByIndex 2, "Texas"
dataTable.addRow newRow
```

NewRow

Creates a new DataRow in the DataTable.

Syntax

```
Function newRow() As DataRow
```

Parameters

- None

Results

Returns the newly created DataRow

Example

```
Dim dataTable As G1CLIENTLib.dataTable
Dim newRow As G1CLIENTlib.DataRow

Set dataTable=requestMsg.getDataTable

Set newRow=dataTable.newRow
newRow.setByName "AddressLine1","10535 Boyer"
newRow.setByName "City", "Austin"
newRow.setByName "State", "Texas"
dataTable.addRow newRow
```

GetRowCount

Gets the number of DataRows in the DataTable.

Syntax

```
Function getRowCount() As Integer
```

Parameters

- None

Results

Returns the number of DataRows in the DataTable.

Example

```
Dim nRowCount As Integer
nRowCount = dataTable.getRowCount
```

Merge

Merges the given DataTable and the current DataTable.

Syntax

```
Sub merge(other As DataTable)
```

Parameters

- The other DataTable to be merged with the current DataTable

Results

None.

Example

```
Dim otherDataTable As New G1CLIENTlib.dataTable
...
dataTable.merge(otherDataTable)
```

DataRow

DataRow contains the individual records for your input and output data. Using the methods associated with this class, you define the column names for your output and add records to the DataTable.

The following table summarizes the functions each method performs in the DataRow class.

Table 37: DataRow Methods Summary

Method	Function
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
getByIndex	Gets the value from the field array by the column index in this DataRow.
getByName	Gets the value from the field array by the column name in this DataRow.
merge	Merges the given DataRow and the current DataRow.
setByName	Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Method	Function
setByIndex	Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.
addChild	Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow Collection, otherwise a new Collection will be created with the supplied DataRow as its only element.
getChildren	Retrieves the child rows from a named relationship.
listChildNames	Retrieves all of the names of the named parent/child relationships.
setChildren	Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

These methods are discussed in more detail in the following sections.

GetColumnNames

Gets all the column names.

Syntax

```
Function getColumnNames() As String()
```

Parameters

- None

Results

Returns the array of column names.

Example

```
Dim sColumnNames() As String
Dim sColumnName As String
Dim nColumn As Integer
ReDim sColumnNames(dataRow.getColumnCount) As String
sColumnName = sColumnNames(nColumn)
For nColumn = 0 To dataRow.getColumnCount -1
```

```
sColumnName = sColumnNames(nColumn)
Next
```

GetColumnIndex

Gets the corresponding column index.

Syntax

```
Function getColumnIndex(columnName As String) As Integer
```

Parameters

- Column name

Results

Returns the corresponding column index.

Example

```
Dim nIndex As Integer
nIndex = dataRow.getColumnIndex("AddressLine1")
```

GetColumnCount

Gets the number of columns in the DataRow.

Syntax

```
Function getColumnCount() As Integer
```

Parameters

- None

Results

Returns the number of columns.

Example

```
Dim nColumnCount As Integer
nColumnCount = dataRow.getColumnCount()
```

GetByIndex

Gets the value from the field array by the column index in this DataRow.

Syntax

```
Function getByIndex(index As Integer) As String
```

Parameters

- Index with which the specified value is to be associated

Results

Returns the value for the column index in this DataRow. Returns empty string if the index is invalid.

Example

```
Dim sValue As String
sValue = dataRow.getByIndex(1)
```

GetByName

Gets the value from the field array by the column name in this DataRow.

Syntax

```
Function getByName(columnName As String) As String
```

Parameters

- Name with which the specified value is to be associated

Results

Returns the value for the column name in this DataRow; returns empty string if the column name does not exist.

Example

```
Dim sValue As String
sValue = dataRow.getByName("City")
```

Merge

Merges the given DataRow and the current DataRow.

Syntax

```
Sub merge(other As DataRow)
```

Parameters

- The other DataRow to be merged with the current DataRow

Results

None.

Example

```
Dim otherDataRow As New G1CLIENTlib.DataRow
...
dataRow.merge(otherDataRow)
```

SetByName

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

```
Sub setByName(columnName As String, value As String)
```

Parameters

- The name with which the specified value is to be associated
- Value to be associated with the specified name

Results

None.

Exceptions

- Blank column name
- Duplicate column name

Example

```
Dim newRow As G1CLIENTLib.DataRow
Set newRow= dataTable.netRow
newRow.setByName "AddressLine1", "100 Congress"
newRow.setByName "City", "Austin"
newRow.setByName "State", "Texas"
dataTable.addRow newRow
```

SetByIndex

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

```
Sub setByIndex(index As Integer, value As String)
```

Parameters

- The column index with which the specified value is to be associated
- Value to be associated with the specified name

Results

None.

Exceptions

- The column index is invalid

Example

```
Dim newRow As G1CLIENTLib.DataRow
Set newRow= dataTable.netRow
newRow.setByIndex 0, "100 Congress"
newRow.setByIndex 1, "Austin"
newRow.setByIndex 2, "Texas"
dataTable.addRow newRow
```

AddChild

Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow collection. Otherwise, a new collection will be created with the supplied DataRow as its only element.

Syntax

```
Sub addChild( childName As String, childDataRow As DataRow)
```

Parameters

- The name of the parent/child relationship (e.g., "Flood Plain Data," "References, " "Used By," etc.)
- The DataRow to be added to the relationship.

Results

None.

Example

```
Dim dataRow As New G1CLIENTLib.dataRow
Dim childDataRow As New G1CLIENTLib.dataRow

childDataRow .setByName "Address", "100 Congress"
childDataRow .setByName "City", "Austin"

dataRow.addChild "child1", dataRow
```

GetChildren

Retrieves the child rows from a named relationship.

Syntax

Function getChildren(childName As String) As DataRow()

Parameters

- The name of the parent/child relationship, e.g. "Flood Plain Data", "References", "Used By", etc.

Result

Returns the child rows from the named relationship.

Example

```
Dim dataRow As New G1CLIENTLib.dataRow
' Assume that dataRow has children ......
' Or more code to be needed
Dim rowsChild1() As Variant
rowsChild1 = dataRow.getChildren("child1")
```

ListChildNames

Retrieves all of the names of the named parent/child relationships.

Syntax

```
Function listChildNames() As String()
```

Parameters

None.

Results

Returns the set of the names of the named parent/child relationships.

Example

```
Dim dataRow As New G1CLIENTLib.dataRow
' Assume that dataRow has children ......
' Or more code to be needed
Dim sChildNames() As String
sChildNames = dataRow.listChildNames
```

SetChildren

Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

Syntax

```
Function setChildren(childName As String, DataRows As DataRow()) As DataRow()
```

Parameters

None.

Results

Returns the set of the names of the named parent/child relationships.

Example

```
Dim dataRow1 As New G1CLIENTLib.dataRow
Dim dataRow2 As New G1CLIENTLib.dataRow
dataRow1.setByName "Address", "100 Congress"
dataRow1.setByName "City", "Austin"
dataRow2.setByName "Address", "200 Congress"
dataRow2.setByName "City", "Austin"

Dim rows(1) As G1CLIENTLib.dataRow

Set rows(0) = dataRow1
Set rows(1) = dataRow2

Dim newRows() As Variant
newRows = dataRowSpt.setChildren("child1", rows())
```

Map

Map is an object that maps keys to values. A map cannot contain duplicate keys — each key can map to at most one value.

The following table summarizes the functions each method performs in the Map class.

Table 38: Map Methods Summary

Method	Function
reset	Sets the cursor to be before the first map.
next	Moves the cursor down one map from its current position.
getKey	Gets the key in the current map.
getValue	Gets the value in the current map.

These methods are discussed in more detail in the following sections.

Reset

Sets the cursor to be before the first map.

Syntax

Sub Reset()

Parameters

- None

Results

None.

Example

```
Dim requestMsg As New G1CLIENTLib.Message
Dim map As G1CLIENTLib.Map
Dim sKey As String
Dim sValue As String

requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_ID, "admin"
requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_PASSWORD, "admin"

Set map = requestMsg.getContextMap

map.Reset
While (map.Next)
 sKey = map.getKey
 sValue = map.getValue
Wend
```

Next

Moves the cursor down one map from its current position.

Syntax

```
Sub Next()
```

Parameters

- None

Example

```
Dim requestMsg As New G1CLIENTLib.Message
Dim map As G1CLIENTLib.Map
Dim sKey As String
Dim sValue As String

requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_ID, "admin"
requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_PASSWORD, "admin"

Set map = requestMsg.getContextMap

map.Reset
While (map.Next)
 sKey = map.getKey
```

```
 sValue = map.getValue
Wend
```

GetKey

Gets the key in the current map.

Syntax

```
Function getKey() As String
```

Parameters

- None

Results

Returns the key on the current map.

Example

```
Dim requestMsg As New G1CLIENTLib.Message
Dim map As G1CLIENTLib.Map
Dim sKey As String
Dim sValue As String

requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_ID, "admin"
requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_PASSWORD, "admin"

Set map = requestMsg.getContextMap

map.Reset
While (map.Next)
 sKey = map.getKey
 sValue = map.getValue
Wend
```

GetValue

Gets the value in the current map.

Syntax

```
Function getValue() As String
```

Parameters

- None

Results

Returns the value on the current map.

Example

```
Dim requestMsg As New G1CLIENTLib.Message
Dim map As G1CLIENTLib.Map
Dim sKey As String
Dim sValue As String

requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_ID, "admin"
requestMsg.putContext requestMsg.CONTEXT_ACCOUNT_PASSWORD, "admin"

Set map = requestMsg.getContextMap

map.Reset
While (map.Next)
 sKey = map.getKey
 sValue = map.getValue
Wend
```

7

The Java API

In this section:

- Introduction 182
- Server 185
- Service 191
- Message 192
- DataTable 198
- DataRow 203

Introduction

A Java class is a blueprint or prototype that defines the variables and methods common to all objects of a certain type. A Java class also defines the implementation of a particular kind of object. It is with these classes that you can create Java applications. In general, Java objects are created from Java classes.

A Java object is a collection of related variables and methods written in the Java language utilizing the Java Virtual Machine (JVM). The data associated with a class or object is stored in variables. The behavior associated with a class or object is implemented with methods. Methods are similar to the functions or procedures in procedural languages such as C.

Java software objects interact and communicate with each other using messages. Additional information that the receiving object may need to perform its task is passed by parameters.

For more information on Java technology, go to **www.sun.com/software/java**.

Constants

The Java API uses two sets of constants. The first set is for the Server component, described in the table below.

Table 39: Constants for the Server Component

Constant Name	Description	Example
Server.HOST	String for server host name. Default is "localhost".	65.89.200.89
Server.PORT	String for server port. Default is "8080".	10119
Server.ACCOUNT_ID	String for server account ID. Default is null.	user1
Server.ACCOUNT_PASSWORD	String for server account password. Default is null.	user1
Server.CONNECTION_TIMEOUT	String for server connection timeout, in millisecond. Default is "10000".	50000
Server.CONNECTION_TYPE	String for server connection type. Currently only support HTTP, HTTPS, or SOCKET. Default is "HTTP".	HTTP
Server.PROXY_HOST	String for proxy server host name. Default is null.	192.168.1.77

Constant Name	Description	Example
Server.PROXY_PORT	String for proxy server port. Default is null.	8080
Server.PROXY_USER	String for proxy server account ID. Default is null.	user1
Server.PROXY_PASSWORD	String for proxy server account password. Default is null.	user1

The second set of constants is for the Message component.

Table 40: Constants for the Message Component

Constant Name	Description/Default	Example
Message.CONTEXT_ACCOUNT_ID	String for message context account ID.	user1
Message.CONTEXT_ACCOUNT_PASSWORD	String for message context account password.	user1
Message.CONTEXT_SERVICE_NAME	String for message context service name.	echoservice

Error Messages

The Java API uses the following error messages:

- Error Messages for Connection
- "Connection type not supported."
- "Client timeout"
- Error Messages for creating DataTable:
- "Blank column name"
- "Duplicated column name"
- "Index is out of bounds"
- Error Messages for Message Packaging Exception
- "Cannot pack null Message"
- "Input Message is null"
- "Unable to connect to Server:"
- "Failed to get Service"
- "Unknown serialization type:"

- "Unknown encoding type:"
- "Gateway is not connected" (for SOCKET)

Example Application

The sample code shown below illustrates how to use the Java API.

```
try
{

    // Create Server
    Server server = new Server();

    // Set server connection properties
    server.setConnectionProperty(Server.HOST, "localhost");
    server.setConnectionProperty(Server.PORT, "10119");
    server.setConnectionProperty(Server.CONNECTION_TYPE, "SOCKET");
    server.setConnectionProperty(Server.ACCOUNT_ID, "guest");
    server.setConnectionProperty(Server.ACCOUNT_PASSWORD, "");

    // Connect to server
    server.connect();

    // Get Service From Server
    Service service = server.getService("ValidateAddress");

    // Create Input Message
    Message request = new Message();

    // Fill DataTable in the input message
    DataTable dataTable = request.getDataTable();
    DataRow row1 = dataTable.newRow();
    row1.set("AddressLine1", "4200 Parliament Place");
    row1.set("City", "Lanham");
    row1.set("StateProvince", "Maryland");
    dataTable.addRow(row1);
    DataRow row2 = dataTable.newRow();
    row2.set("AddressLine1", "100 Congress");
    row2.set("City", "Austin");
    row2.set("StateProvince", "Texas");
    dataTable.addRow(row2);

    // Set"option" Properties to the Input
    Message request.putOption("OutputCasing", "M");
    request.putOption("OutputRecordType", "A");

    // Process Input Message, return output Message
    Message reply = service.process(request);

    // Disconnect from server
    server.disconnect();

    // Get the result from the response message
    DataTable returnDataTable = reply.getDataTable();
    String[] columnNames = returnDataTable.getColumnNames();
    Iterator iter = returnDataTable.iterator();
    while (iter.hasNext())
    {
        DataRow row = (DataRow) iter.next();
```

```
        for (int col = 0; col & lt; returnDataTable.getColumnCount(); col++)

        {
            String value = row.get(columnNames[col]);
            System.out.println(value);
        }
    }
}
catch (Exception e)
{
    System.out.println("Error Occurred, " + e.getMessage());
}
```

Server

The Server class is used to connect to the server, disconnect from the server, and get the service from the server. The following table summarizes the functions each method performs in the Server class.

Table 41: Server Methods Summary

Method	Function
connect	Connects to the server.
disconnect	Disconnects from the server.
setConnectionProperty	Sets the configuration items for the connection to the server.
getService	Gets the service from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Connect

Reads the properties to determine which gateway connection to be used and makes a connection to the server. You can connect via HTTP, HTTPS, or SOCKET. However, HTTP and HTTPS do not actually connect to the server until a GetService or Process method is invoked. With a SOCKET connection type, the Connect method is fully functional.

Syntax

```
public void connect()
```

Parameters

None.

Results

Throws:

- ConfigurationException: When invalid configuration causes the inability to connect to the server. For example, an unknown protocol would cause a ConfigurationException. There is no value in attempting to retry connect() when this error occurs.
- ConnectionException: When unable to connect to the server. It might be possible to reconnect, depending on the underlying cause of the exception.
- MessageProcessingException: When an error occurs on the server that is not due to Configuration or Connection issues.

Example

```
Server server = new Server();

server.setConnectionProperty(Server.HOST, "localhost");
server.setConnectionProperty(Server.PORT, "10119");
server.setConnectionProperty(Server.CONNECTION_TYPE, "SOCKET");
server.setConnectionProperty(Server.ACCOUNT_ID, "guest");
server.setConnectionProperty(Server.ACCOUNT_PASSWORD, "");

try
{
 //Connect to server
 server.connect();
}
catch (ConfgurationException e)
{
 // indicate an error with configuration
}
catch (ConnectionException e)
{
 // handle connection issue (retry, report error, etc.)
}
catch (MessageProcessingException e)
{
 // report error
}
```

Connection Pooling

Connection pooling for the SOCKET connection type is available to the Java client. This section describes how to enable and disable connection pooling. By default connection pooling is disabled.

To enable connection pooling:

```
Server server = new Server();
Server.setConnectionProperty(Connection.SOCKET_POOL, "true");
```

To disable connection pooling:

```
Server server = new Server();
Server.setConnectionProperty(Connection.SOCKET_POOL, "false");
```

When connection pooling is enabled, the connect() method borrows a connection from the pool, and the disconnect() method returns the connection back to the pool. When pooling, the client must call disconnect() each time to return the connection to the pool.

Each thread should contain its own server, as shown in the following example:

```
{
 ...
 Server server = new Server();
 server.setConnectionProperty(Server.HOST, "localhost");
 server.setConnectionProperty(Server.PORT, "10119");
 server.setConnectionProperty(Server.CONNECTION_TYPE, "SOCKET");
 server.setConnectionProperty(Server.ACCOUNT_ID, "yourID");
 server.setConnectionProperty(Server.ACCOUNT_PASSWORD, "pwd");
 server.setConnectionProperty(Connection.SOCKET_POOL, "true");
 server.setConnectionProperty(Connection.SOCKET_POOL_MAX_ACTIVE, "20");
 server.setConnectionProperty(Connection.SOCKET_POOL_MIN_IDLE, "10");
 server.setConnectionProperty(Connection.SOCKET_POOL_MAX_TOTAL, "25");
 server.connect();
 ...
 service = server.getService(serviceName);
 reply = service.process(requestMessage);
 server.disconnect();
 ...
}
```

The following table lists the constants you can use for connection pooling.

Table 42: Constants for Connection Pooling

Constant Name	Description
SOCKET_POOL	Whether or not to use connection pooling if using the SOCKET connection type. Valid values are true or false. Default is false.
SOCKET_POOL_MAX_ACTIVE*	Maximum number of active socket connections that may be borrowed from the pool. Default is -1, which indicates no maximum.
SOCKET_POOL_MAX_IDLE*	Maximum number of idle socket connections remaining in the pool. Default is -1, which indicates no maximum.
SOCKET_POOL_MAX_TOTAL*	Maximum total number of pooled socket connections (both active and idle). Default is -1, which indicates no maximum.

Constant Name	Description
SOCKET_POOL_MAX_WAIT*	Maximum amount of time (in milliseconds) to wait before throwing an exception when the pool is exhausted and the "when exhausted" action is WHEN_EXHAUSTED_BLOCK. Default is -1, which indicates no maximum.
SOCKET_POOL_MIN_EVICTABLE_IDLE_TIME_MILLIS*	Minimum amount of time a connection may sit idle in the pool before it is eligible for eviction. Default is 1800000 (30 minutes).
SOCKET_POOL_MIN_IDLE*	Minimum number of connections allowed in the pool before the evictor thread (if active) creates new connections. Default is 0.
SOCKET_POOL_NUM_TESTS_PER_EVICTION_RUN*	Sets the number of idle connections to examine during each run of the evictor thread (if active). Default is -1, which indicates all idle connections are examined.
SOCKET_POOL_TEST_ON_BORROW*	Whether connections will be validated before being borrowed from the pool. Default is true.
SOCKET_POOL_TEST_ON_RETURN*	Whether connections will be validated before being returned to the pool. Default is false.
SOCKET_POOL_TEST_WHILE_IDLE*	Whether connections will be validated by the idle connection eviction thread. Default is false.
SOCKET_POOL_TIME_BETWEEN_EVICTION_RUNS_MILLIS*	Sets the number of milliseconds to sleep between runs of the idle connection evictor thread. When set to zero or a negative number, no idle connection evictor thread will be run. Default is 300000 (5 minutes).

Constant Name	Description
SOCKET_POOL_WHEN_EXHAUSTED_ACTION*	Sets the "when exhausted action" to take when attempting to borrow a connection and none are available. Default is SOCKET_POOL_WHEN_EXHAUSTED_BLOCK.
SOCKET_POOL_WHEN_EXHAUSTED_BLOCK*	A "when exhausted action" type indicating that when attempting to borrow a connection and none are available, the caller should block until a new object is available, or the maximum wait time has elapsed.
SOCKET_POOL_WHEN_EXHAUSTED_FAIL*	A "when exhausted action" type indicating that when attempting to borrow a connection and none are available, the caller should fail, throwing a ConnectionException.
SOCKET_POOL_WHEN_EXHAUSTED_GROW*	A "when exhausted action" type indicating that when attempting to borrow a connection and none are available, a new connection will be made anyway.

* Applicable only if using the SOCKET connection type and connection pooling is enabled.

Disconnect

Disconnects from the server.

Syntax

```
public void disconnect()
```

Parameters

None.

Results

Client is disconnected from the server.

Example

```
...
//Disconnect from server
server.disconnect();
```

SetConnectionProperty

Establishes the server connection configuration properties, such as host name and length of timeout.

Syntax

```
public void setConnectionProperty(String name, String value)
```

Parameters

- Name — the name of the connection property, such as HOST
- Value — the value for the name of the connection property, such as "www.myhost.com"

Results

None.

Exceptions

- ERROR-INVALID-COLUMN_NAME — an empty of null column name.
- ERROR_INVALID_VALUE — A null value.

Example

```
Server server = new Server();

server.setConnectionProperty(Server.HOST, "localhost");
server.setConnectionProperty(Server.PORT, "8080");

//Connect to server
server.connect();
```

GetService

Gets the service from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Syntax

```
public Service getService(String serviceName)
```

Parameters

- Name - the name of the service

Results

Returns the specific service.

Exceptions

throws ServiceNotFoundException, ServiceCreationException

Example

```
Service service = server.getService("ValidateAddress");
```

Service

The Service class is used to process the message (i.e., send the message to the server and receive a response from the server).

This class has just one method: Process. The Process method is detailed below.

Process

Process the input message and returns the response message.

Syntax

```
public Message process (Message message)
```

Parameters

- Input message

Results

Returns the response message.

Exceptions

- TimeoutException: When invalid configuration causes the inability to connect to the server. For example, an unknown protocol would cause a ConfigurationException. There is no value in attempting to retry connect() when this error occurs.
- ConnectionException: When unable to connect to the server. It might be possible to reconnect, depending on the underlying cause of the exception.
- MessageProcessingException: When an error occurs on the server that is not due to Configuration or Connection issues.

Example

```
try
{
```

```
 //Process Input Message, return output Message
 Message response = service.process(message);
}
catch (ConnectionException e)
{
 // handle connection issue (retry, report error, etc.)
}
catch (TimeoutException e)
{
 // handle timeout issue (retry, report error, etc.)
}
catch (MessageProcessingException e)
{
 // report error
}
```

Message

The Message class sends your input data and receives your output data from the service. The properties for Message include context properties, such as account ID, account password, service name, and service method; and option properties, which are the service-specific runtime options.

The following table summarizes the functions each method performs in the Message class.

Table 43: Message Methods Summary

Method	Function
getContext	Gets the value of the context entity identified by name in the context section of the message.
getContext	Gets the map that contains all of the context entries.
putContext	Sets the value of the context entity identified by name in the context session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putContext	Add the new context properties to the current context properties.
setContext	Overwrites the current context properties with the new context properties.
getOption	Gets the value of the option entity identified by name in the option section of the message.

Method	Function
getOptions	Gets the map that contains all of the option entries.
putOption	Sets the value of the option entity identified by name in the option section of the message. If there is an existing value present for the entity identified by the name, it is replaced.
putOptions	Add the new option properties to the current option properties.
setOptions	Overwrites the current option properties with the new option properties.
getError	Gets the error from the error message.
getDataTable	Gets the specified data table from the message.

GetContext

Gets the value by the name in the "context" properties. Context properties include the following constants: account ID, account password, service name, service key, and request ID.

Syntax

```
public String getContext(String name)
```

Parameters

- Name - the name whose associated value is to be returned

Results

Returns the value associated with the name in the context properties. If the name does not exist, the method returns NULL.

Example

```
String value = message.getContext(Message.CONTEXT_ACCOUNT_ID);
```

GetContext

Gets the map that contains all of the context entries.

Syntax

```
public Map getContext()
```

Parameters

- None

Results

Returns the map that contains all of the context entries.

Example

```
Map context = message.getContext();
```

PutContext

Sets the value for the given name in the context properties. If there is an existing value present for the entity identified by the name, it is replaced. Context properties include the following constants: accountID, account password, service name, service key, and request ID.

Syntax

```
public void putContext(String name, String value)
```

Parameters

- Name - the name with which the specified value is to be associated
- Value - the value to be associated with the specified name

Results

None.

Example

```
message.putContext(Message.CONTEXT_ACCOUNT_ID, "user1");
```

PutContext

Adds the new context properties to the current context properties.

Syntax

```
public void putContext(Map map)
```

Parameters

- The new context hashtable to be added to the current context hashtable

Results

None.

Example

```
Map context = new HashMap();
...
message.putContext(context);
```

SetContext

Overwrites the current context properties with the new context properties.

Syntax

```
public void setContext(Map map)
```

Parameters

- The new context map that will replace the current context map.

Results

None.

Example

```
Map context = new Map ();
...
message.setContext(context);
```

GetOption

Gets the value by the name in the option properties. Option properties are the service-specific run-time options.

Syntax

```
public String getOption(String name)
```

Parameters

- Name - the name whose associated value is to be returned.

Results

Returns the value for the name in the option properties in the message OR NULL if the name does not exist.

Example

```
String value = message.getOption("OutputCasing");
```

GetOptions

Gets the map that contains all of the option entries.

Syntax

```
public Map getOptions();
```

Parameters

- None

Results

Returns the map that contains all of the option entries.

Example

```
Map options = message.getOptions();
```

PutOption

Sets the value for given name in the option properties. If there is an existing value present for the entity identified by the name, it is replaced. Option properties are the service specific run-time options.

Syntax

```
public void setOption(String name, String value)
```

Parameters

- Name - name with which the specified value is to be associated
- Value - value to be associated with the specified name

Results

None.

Example

```
message.setOption("OutputCasing", "M");
```

PutOptions

Adds the new option properties to the current option properties.

Syntax

```
public void putOptions(Map map)
```

Parameters

- The new option map to be added to the current option properties

Example

```
Map options = new HashMap();
...
message.putOptions(options);
```

SetOptions

Overwrites the current option properties with the new option properties.

Syntax

```
public void setOptions(Map map)
```

Parameters

- The new option map to replace the current option map

Results

None.

Example

```
Map options = new HashMap();
...
message.setOptions(options);
```

GetError

Gets the error message from the message.

Syntax

```
public String getError()
```

Parameters

- None

Results

Returns the error message in the message.

Example

```
String error = message.getError();
```

GetDataTable

Gets the data table in this message.

Syntax

```
public DataTable getDataTable()
```

Parameters

None.

Results

None.

Example

```
DataTable dataTable = message.getDataTable();
```

DataTable

DataTable contains the records for your input and output data. Using the methods associated with this class, you define the column names for your output and add records to the DataTable.

The following table summarizes the functions each method performs in the DataTable class.

Table 44: DataTable Methods Summary

Method	Function
addColumn	Adds a new column to the DataTable.
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
getColumnCount	Gets the number of columns.
clear	Clears all the data in the DataTable.
iterator	An iterator that contains all the rows in the DataTable.
addRow	Adds a DataRow to the DataTable.

Method	Function
newRow	Creates a new DataRow in the DataTable.
getRowCount	Gets the number of DataRows in the DataTable.
merge	Merge the given DataTable and the current DataTable.

These methods are discussed in more detail in the following sections.

AddColumn

Adds the new column to the DataTable.

Syntax

```
public int addColumn(String columnName)
```

Parameters

- columnName

Results

Returns the index of the column

Example

```
DataTable dataTable = message.getDataTable();
int columnIndex = dataTable.addColumn("AddressLine1");
columnIndex = dataTable.addColumn("City")
```

GetColumnNames

Gets all the column names.

Syntax

```
public String[] getColumnNames()
```

Parameters

- None

Results

Returns the string array of column names.

Example

```
String[] columnNames = dataTable.getColumnNames();
```

GetColumnIndex

Gets the corresponding column index.

Syntax

```
public int getColumnIndex(String columnName)
```

Parameters

- Column name

Results

Returns the corresponding column index.

Example

```
int columnIndex = dataTable.getColumnIndex("City");
```

GetColumnCount

Gets the number of columns in the DataTable.

Syntax

```
public int getColumnCount()
```

Parameters

- None

Results

Returns the number of columns.

Example

```
int columnCount = dataTable.getColumnCount();
```

Clear

Clears the data in the DataTable.

Syntax

```
public void clear()
```

Parameters

- None

Results

None.

Example

```
dataTable.clear();
```

Iterator

An iterator that contains all DataRows in the DataTable.

Syntax

```
public Iterator iterator()
```

Parameters

- None

Results

Returns an iterator that contains all DataRows in the DataTable.

Example

```
Iterator iter = dataTable.iterator();
while (iter.hasNext())
{
 DataRow row = (DataRow)iter.next();
}
```

AddRow

Adds a row to the DataTable.

Syntax

```
public void addRow(DataRow row)
```

Parameters

- Row - DataRow to be added to the DataTable

Results

None.

Example

```
DataTable dataTable = message.getDataTable();

DataRow row = dataTable.newRow();
row.set("AddressLine1","4203 Greenridge");

dataTable.addRow(row);
```

NewRow

Creates a new DataRow to the DataTable.

Syntax

```
public DataRow newRow()
```

Parameters

- None

Results

Returns the newly created DataRow

Example

```
DataRow row = dataTable.newRow();
row.set("AddressLine1", "4203 Greenridge");

dataTable.addRow(row);
```

GetRowCount

Gets the number of DataRows in the DataTable.

Syntax

```
public int getRowCount()
```

Parameters

- None

Results

Returns the number of DataRows in the DataTable.

Example

```
int rowCount = dataTable.getRowCount();
```

Merge

Merges the given DataTable and the current DataTable.

Syntax

```
public void merge(DataTable other)
```

Parameters

- The other DataTable to be merged with the current DataTable

Results

None.

Example

```
DataTable otherDataTable = new DataTable();
dataTable.merge(otherDataTable);
```

DataRow

DataRow contains the individual records for your input and output data. Using the methods associated with this class, you define the column names for your output and add records to the DataTable.

The following table summarizes the functions each method performs in the DataRow class.

Table 45: DataRow Methods Summary

Method	Function
getColumnNames	Gets all the column names.
getColumnIndex	Gets the corresponding column index.
get	Gets the value from the field array by the column index in this DataRow.
get	Gets the value from the field array by the column name in this dataTow.
merge	Merges the given DataRow and the current DataRow.

Method	Function
set	Sets the value for the corresponding column name for the DataRow. If the value for the name exists, the old value is replaced.
set	Sets the value for the corresponding column index for the DataRow. If the value for the name exists, the old value is replaced.
addChild	Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow Collection, otherwise a new Collection will be created with the supplied DataRow as its only element.
getChildren	Retrieves the child rows from a named relationship.
listChildNames	Retrieves all of the names of the named parent/child relationships
setChildren	Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

These methods are discussed in more detail in the following sections.

GetColumnNames

Gets all the column names.

Syntax

```
public String[] getColumnNames()
```

Parameters

- None

Results

Returns the string array of column names.

Example

```
String[] columnNames = dataRow.getColumnNames();
```

GetColumnIndex

Gets the corresponding column index.

Syntax

```
public int getColumnIndex(String columnName)
```

Parameters

- Name - column name

Results

Returns the corresponding column index.

Example

```
int columnIndex = dataRow.getColumnIndex("City");
```

Get

Gets the value from the field array by the column index in this DataRow.

Syntax

```
public String get(int index)
```

Parameters

- Index with which the specified value is to be associated

Results

Returns the value for the column index in this DataRow.

Example

```
String value = dataRow.get(1);
```

Get

Gets the value from the field array by the column name in this DataRow.

Syntax

```
public String get(String columnName)
```

Parameters

- Name - name with which the specified value is to be associated

Results

Returns the value for the column name in this DataRow; returns empty string if the column name does not exist.

Example

```
String value = dataRow.get("City");
```

Merge

Merges the given DataRow and the current DataRow.

Syntax

```
public void merge(DataRow other)
```

Parameters

- The other DataRow to be merged with the current DataRow

Results

None.

Example

```
DataRow otherDataRow = new DataRow();
dataRow.merge(otherDataRow);
```

Set

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

```
public void set(int Index, String value)
```

Parameters

- The column index with which the specified value is to be associated
- Value to be associated with the specified name

Results

None.

Exceptions

- IndexOutOfBounds — the column index is invalid

Example

```
DataRow row = dataTable.newRow();
row.set(0,"4203 Greenridge");
row.set(1,"Austin");
row.set(2,"Texas")
dataTable.addRow(row);
```

AddChild

Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow collection. Otherwise, a new collection will be created with the supplied DataRow as its only element.

Syntax

```
public void addChild(String childName, DataRow childDataRow)
```

Parameters

- Name - the name of the parent/child relationship (e.g., "Flood Plain Data," "References, " "Used By," etc.)
- Value - the DataRow to be added to the relationship.

Results

None.

Example

```
DataRow childDataRow = new DataRow();
childDataRow.set("Address", "100 Congress");
...
DataRow dataRow = new DataRow();
...
dataRow.addChild("child1", childDataRow);
```

GetChildren

Retrieves the child rows from a named relationship.

Syntax

```
public List getChildren(String childName)
```

Parameters

- The name of the parent/child relationship, e.g. "Flood Plain Data", "References", "Used By", etc.

Results

Returns the child rows from the named relationship.

Example

```
List childRows = row.getChildren("child1");
```

ListChildNames

Retrieves all of the names of the named parent/child relationships.

Syntax

```
public Set listChildNames()
```

Parameters

None.

Results

Returns the set of the names of the named parent/child relationships.

Example

```
Set childNames = row.listChildNames();
```

SetChildren

Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

Syntax

```
public List setChildren(String childName, List DataRows)
```

Parameters

None.

Results

Returns the set of the names of the named parent/child relationships.

Example

```
List rows = dataRow.getChildren("child1");
parentRow.setChildren("child2", rows);
```

Set

Sets the value for the corresponding column for the DataRow. If the value for the name exists, the old value is replaced.

Syntax

```
public void set(int Index, String value)
```

Parameters

- The column index with which the specified value is to be associated
- Value to be associated with the specified name

Results

None.

Exceptions

- IndexOutOfBounds — the column index is invalid

Example

```
DataRow row = dataTable.newRow();
row.set(0,"4203 Greenridge");
row.set(1,"Austin");
row.set(2,"Texas")
dataTable.addRow(row);
```

8

ManagementAPI Methods

In this section:

- Introduction 212
- GetLicenseInfo 212
- GetVersionInfo 213

Introduction

There are two management API methods that are publicly available via the ManagementAPI Web service. Those methods include getLicenseInfo and getVersionInfo. The wsdl for the ManagementAPI Web service can be viewed using the following URL: `http://localhost:8080/managers/ManagementAPIService?wsdl`, where "localhost" is replaced by the name of the Spectrum™ Technology Platform server.

GetLicenseInfo

The method GetLicenseInfo returns a license object. The license object contains properties for machine type, operating system type, hostname, and CPU limit. It also contains an array of feature objects and an array of restriction objects. These arrays can be processed to determine specific information about features and restrictions. Feature has an ID, a name, and an enabled flag. Restriction has an ID, a limit, and a start date.

Web Service

ManagementAPIService

Parameters

None.

Result

Returns the license object.

Example

```
License
 string machineType
 string osType
 string hostName
 string CPULimit
 Feature[] features
 Restriction[] restrictions

Feature
 string ID;
 string name;
 Restriction[] restrictions

Restriction
 string ID
 long limit
 datetime startDate
 Feature[] features
```

```
ExpirationRestriction extends Restriction

UsageRestriction extends Restriction
 long usages
```

GetVersionInfo

The GetVersionInfo method returns an array of VersionInfo objects. A VersionInfo object has a name, version number, and a list of VersionAttribute objects. VersionAttribute objects are simple classes that have a label and a value. GetVersionInfo attributes are product-specific as the information is gathered and returned by the product itself. This same information is also displayed in the Version Information node of the Management Console.

Note: You must run GetVersionInfo once, see what values come back, and then parse the information to get specific pieces of information.

Web Service

ManagementAPIService

Parameters

None.

Result

Returns VersionInfo objects.

Example

```
VersionInfo
 string name
 string version
 VersionAttribute[] attributes

VersionAttribute
 string label
 string value
```

9

The .NET API

In this section:

- Introduction 216
- Server 219
- Service 222
- Message 223
- EnhancedDataTable 229

Introduction

.NET is a Microsoft® operating system platform that incorporates applications and a suite of tools and services which enhance Web service and application development.

The .NET framework uses components called Common Language Runtime (CLR), Framework Class Library (FCL), and ASP.NET. The CLR is equivalent to the Java Virtual Machine, in that it manages code and executes it in the native language of the machine on which it runs. The Framework Class Library is a massive library of re-usable object types that cover a myriad of program functions. ASP.NET is a server-side technology that allows web pages and services to load much faster than traditional ASP pages. Together, these three components of the .NET framework make application and Web development easier, more streamlined, and provides easier integration into existing environments. Clients and servers on different platforms running services written in various programming languages can communicate with each other swiftly and easily.

For more information on .NET technology, go to **msdn.microsoft.com/netframework**.

Constants

The .NET API uses two sets of constants. The first set is for the Server component, described in the table below.

Table 46: Constants for the Server Component

Constant Name	Description	Example
Server.HOST	String for server host name. Default is "localhost".	65.89.200.89
Server.PORT	String for server port. Default is "8080".	10119
Server.ACCOUNT_ID	String for server account ID. Default is null.	user1
Server.ACCOUNT_PASSWORD	String for server account password. Default is null.	user1
Server.CONNECTION_TIMEOUT	String for server connection timeout, in millisecond. Default is "10000".	50000
Server.CONNECTION_TYPE	String for server connection type. Currently only support HTTP, HTTPS, or SOCKET. Defaults is "HTTP".	HTTP(S)

Constant Name	Description	Example
Server.PROXY_HOST	String for proxy server host name. Default is null.	192.168.1.77
Server.PROXY_PORT	String for proxy server port. Default is null.	8080
Server.PROXY_USER	String for proxy server account ID. Default is null.	user1
Server.PROXY_PASSWORD	String for proxy server account password. Default is null.	user1

The second set of constants is for the Message component.

Table 47: Constants for the Message Component

Constant Name	Description	Example
Message.CONTEXT_ACCOUNT_ID	String for message context account ID.	user1
Message.CONTEXT_ACCOUNT_PASSWORD	String for message context account password.	user1
Message.CONTEXT_SERVICE_NAME	String for message context service name.	echoservice

Error Messages

The .NET API uses the following error messages:

- Error Messages for Connection
- "Connection type not supported."
- "Client timeout"
- Error Messages for Message Packaging Exception
- "Input Message is null."

The other error messages will come from the .NET Framework Class Library if they do not get used properly.

Example Application

The sample code shown below illustrates how to use the .NET API.

```
using System;
using System.IO;
using System.Collections;
using System.Text;
using System.Data;

try
 {
 //Create Server
 Server server = new Server();

 //Set connect property to the server
 server.SetConnectionProperty(Server.HOST, "localhost");
 server.SetConnectionProperty(Server.PORT, "10119");
 server.SetConnectionProperty(Server.CONNECTION_TYPE, "SOCKET");
 server.SetConnectionProperty(Server.ACCOUNT_ID, "guest");
 server.SetConnectionProperty(Server.ACCOUNT_PASSWORD, "");

 //Connect to server
 server.Connect();

 //Get Service From Server
 Service service = server.GetService("ValidateAddress");

 //Create Input Message
 Message request = new Message();

 //Fill dataTable in the input message
 //Datatable is the .net Framework class
 DataTable dataTable = request.GetDataTable();

 DataColumn column1 = new DataColumn();
 column1.DataType = System.Type.GetType("System.String");
 column1.ColumnName = "AddressLine1";
 dataTable.Columns.Add(column1);

 DataColumn column2 = new DataColumn();
 column2.DataType = System.Type.GetType("System.String");
 column2.ColumnName = "City";
 dataTable.Columns.Add(column2);

 DataColumn column3 = new DataColumn();
 column3.DataType = System.Type.GetType("System.String");
 column3.ColumnName = "StateProvince";
 dataTable.Columns.Add(column3);

 DataRow newRow = dataTable.NewRow();
 newRow[0]="4200 Parliament Place";
 newRow[1]="Lanham";
 newRow[2]="Maryland";

 dataTable.Rows.Add(newRow);

 //Set "option" Properties to the Input Message
 request.PutOption("OutputCasing", "M");
```

```
request.PutOption("OutputRecordType", "A");

//Process Input Message, return output Message
Message reply = service.Process(request);

//Disconnect from server
server.Disconnect();

//Get the result from the response message
DataTable returnDataTable = reply.GetDataTable();

foreach(DataColumn dc in returnDataTable.Columns)
{
// more code to be added
string columnName = dc.ColumnName;
}
foreach(DataRow dr in returnDataTable.Rows)
{
for (int col = 0; col < returnDataTable.Columns.Count; col++)
{
// more code to be added
string value = (String)dr[col] ;
Console.WriteLine(value);
}
}
}
catch (Exception e)
{

//Error handling
Console.WriteLine("Error Ocurred, " + e.ToString());
}
```

Server

The Server class is used to connect to server, disconnect from the server, and get the service from the server. The following table summarizes the functions each method performs in the Server class.

Table 48: Server Methods Summary

Method	Function
Connect	Connects to the server.
Disconnect	Disconnects from the server.
SetConnectionProperty	Sets the configuration items for the connection to the server.

Method	Function
GetService	Gets the service (for example, ValidateAddress) from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Connect

Reads the properties to determine which gateway connection to be used and makes a connection to the server.

Note: .NET uses the HTTP, HTTPS, or SOCKET server connection protocol. HTTP and HTTPS logically establish a client connection, but do not actually connect to the server until a GetService or Process method is invoked. The SOCKET protocol establishes a connection to the server when Connect is invoked.

Syntax

```
public void Connect()
```

Parameters

None.

Results

None.

Exceptions

- "Connection type not supported."

Example

```
Server server = new Server();

// set connect property to the server
 server.SetConnectionProperty(Server.HOST, "localhost");
server.SetConnectionProperty(Server.PORT, "8080");
// more connection properties to be set
// Connect to server
server.Connect();
```

Disconnect

Disconnects from the server.

Syntax

```
public void Disconnect()
```

Parameters

None.

Results

Client is disconnected from the server.

Example

```
//Disconnect from server
server.Disconnect();
```

SetConnectionProperty

Establishes the server connection configuration properties, such as host name and length of timeout.

Syntax

```
public void SetConnectionProperty(String name, String value)
```

Parameters

- Name — the name of the connection property, such as HOST
- Value — the value for the name of the connection property, such as "www.myhost.com"

Results

None.

Example

```
Server server = new Server();

server.SetConnectionProperty(Server.HOST, "localhost");
server.SetConnectionProperty(Server.PORT, "8080");

//Connect to server
server.Connect();
```

GetService

Gets the service from the server.

Note: See the Component Reference section of this guide for a list of services that may be available to you.

Syntax

```
public Service getService(String serviceName)
```

Parameters

- Name - name of service

Results

Returns the specific service.

Example

```
Service service = server.GetService("ValidateAddress");
```

Service

The Service class is used to process the message (in other words, it sends the message to the server and receives a response from the server).

This class has just one method: Process. The Process method is detailed below.

Process

Processes the input message and returns the response message.

Syntax

```
public Message Process(Message, message)
```

Parameters

- Input message

Results

Returns the response message.

Exceptions

MessageProcessingException

Example

```
//Process Input Message, return output Message
Message reply = service.Process(request);
```

Message

The Message class sends your input data and receives your output data from the service. The properties for Message include context properties, such as account ID, account password, service name, and service method; and option properties, which are the Service-specific runtime options.

The following table summarizes the functions each method performs in the Message class.

Table 49: Message Methods Summary

Method	Function
GetContext	Gets the value of the context entity identified by name in the context section of the message.
GetContext	Gets the properties that contains all of the context entries.
PutContext	Sets the value of the context entity identified by name in the context session of the message. If there is an existing value present for the entity identified by the name, it is replaced.
PutContext	Adds the new context properties to the current context properties.
SetContexts	Overwrites the current context properties with the new context properties.
GetOption	Gets the value of the option entity identified by name in the option section of the message.
GetOptions	Gets the properties that contains all of the option entries.
PutOption	Sets the value of the option entity identified by name in the option section of the message. If there is an existing value present for the entity identified by the name, it is replaced.
PutOptions	Adds the new option properties to the current option properties.

Method	Function
SetOptions	Overwrites the current option properties with the new option properties.
GetError	Gets the error from the error message.
GetDataTable	Gets the specified data table from the message.

GetContext

Gets the value by the name in the context properties. Context properties include the following constants: account ID, account password, service name, service key, and the request ID.

Syntax

```
public String GetContext(String name)
```

Parameters

None.

Results

Returns the value associated with the name in the "context" properties. If the name does not exist, the method returns NULL.

Example

```
String value = message.GetContext(Message.CONTEXT_ACCOUNT_ID);
```

GetContext

Gets the hashtable that contains all of the context entries. Hashtable is the .NET Framework class.

Syntax

```
public Hashtable GetContext()
```

Parameters

- None

Results

Returns the hashtable that contains all of the context entries.

Example

```
Hashtable context = message.GetContext();
```

PutContext

Sets the value for the given name in the context properties. If there is an existing value present for the entity identified by the name, it is replaced. Context properties include the following constants: account ID, account password, service name, service key, and request ID.

Syntax

```
public void PutContext(String name, String value)
```

Parameters

- Name - name with which the specified value is to be associated
- Value - value to be associated with the specified name

Example

```
message.PutContext(Message.CONTEXT_ACCOUNT_ID, "user1");
```

PutContext

Adds the new context properties to the current context properties.

Syntax

```
public void PutContext(Hashtable context)
```

Parameters

- The new context hashtable to be added to the current context hashtable

Results

None.

Example

```
//Hashtable is the .NET Framework class
Hashtable context = new Hashtable();
//more code
message.PutContext(context);
```

SetContexts

Overwrite the current context properties with the new context properties.

Syntax

```
public void SetContexts(Hashtable context)
```

Parameters

- Context - the new context hashtable that will replace the current context hashtable.

Results

None.

Example

```
//Hashtable is the .NET Framework class
Hashtable context = new Hashtable();
//more code
message.SetContexts(context);
```

GetOption

Gets the value by the name in the option properties. Option properties are the service-specific run-time options.

Syntax

```
public String GetOption(String name)
```

Parameters

- Name - the name whose associated value is to be returned.

Results

Returns the value for the name in the "option" properties in the message or NULL if the name does not exist.

Example

```
String value = message.GetOption("OutputCasing");
```

GetOptions

Gets the hashtable that contains all of the option entries. Hashtable is the .NET Framework class.

Syntax

```
public Hashtable GetOptions();
```

Parameters

- None

Results

Returns the hashtable that contains all of the option entries.

Example

```
Hashtable options = message.GetOptions();
```

PutOption

Sets the value for given name in the option properties. If there is an existing value present for the entity identified by the name, it is replaced. Option properties are the Service specific run-time options.

Syntax

```
public void PutOption(String name, String value)
```

Parameters

- Name - name with which the specified value is to be associated
- Value - value to be associated with the specified name

Example

```
message.PutOption("OutputCasing", "M");
```

PutOptions

Adds the new option properties to the current option properties.

Syntax

```
public void PutOptions(Hashtable options)
```

Parameters

- Option - the new option hashtable to be added to the current option hashtable

Results

None.

Example

```
//Hastable is the .NET Framework class
Hashtable options = new Hashtable();
// more code
message.PutOptions(options);
```

SetOptions

Overwrites the current option properties with the new option properties.

Syntax

```
public void SetOptions(Hashtable options)
```

Parameters

- Options - the new option hashtable to replace the current option hashtable

Results

None.

Example

```
//Hashtable is the .NET Framework class
Hashtable options = new Hashtable();
//more code
message.SetOptions(options);
```

GetError

Gets the error message from the message.

Syntax

```
public String GetError()
```

Parameters

- None

Results

Returns the error message in the message.

Example

```
String error = message.GetError();
```

GetDataTable

Gets the DataTable in this message. DataTable is .NET Framework class.

Syntax

```
public DataTable GetDataTable()
```

Parameters

None.

Results

None.

Example

```
//DataTable is the .net Framework class
DataTable dataTable = message.GetDataTable();

DataColumn column1 = new DataColumn();
column1.DataType = System.Type.GetType("System.String");
column1.ColumnName = "AddressLine1";
dataTable.Columns.Add(column1);

DataColumn column2 = new DataColumn();
column2.DataType = System.Type.GetType("System.String");
column2.ColumnName = "City";
dataTable.Columns.Add(column2);

DataRow newRow = dataTable.NewRow();
newRow[0]="4203 Greenridge";
newRow[1]="Austin";

dataTable.Rows.Add(newRow);
```

EnhancedDataTable

EnhancedDataTable is a class which extends .Net class DataTable. The following table summarizes the functions each method performs in the EnhancedDataTable class.

Table 50: EnhancedDataTable Methods Summary

Method	Function
AddChild	Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow Collection, otherwise a new Collection will be created with the supplied DataRow as its only element.
GetChildren	Retrieves the child rows from a named relationship.
ListChildNames	Retrieves all of the names of the named parent/child relationships.

Method	Function
SetChildren	Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

AddChild

Adds a new DataRow to the named parent/child relationship. If the named relationship exists, the supplied DataRow will be appended to the existing DataRow collection. Otherwise, a new collection will be created with the supplied DataRow as its only element.

Syntax

```
public void AddChild(DataRow parentRow, string name, DataRow newChild)
```

Parameters

- Name - the name of the parent/child relationship (e.g., "Flood Plain Data," "References, " "Used By," etc.)
- DataRow - the DataRow to be added to the relationship.

Results

None.

Example

```
EnhancedDataTable dataTable = new EnhancedDataTable();

dataTable.Columns.Add(new DataColumn("AddressLine1",
System.Type.GetType("System.String")));
dataTable.Columns.Add(new DataColumn("City",
System.Type.GetType("System.String")));
dataTable.Columns.Add(new DataColumn("StateProvince",
System.Type.GetType("System.String")));
dataTable.Columns.Add(new DataColumn("PostalCode",
System.Type.GetType("System.String")));

DataRow row = dataTable.NewRow();

row[0] = "510 S Coit St";
row[1] = "Florence";
row[2] = "SC";
row[3] = "29501-5221";

EnhancedDataTable childDataTable = new EnhancedDataTable();

childDataTable.Columns.Add(new DataColumn("AddressLine2",
System.Type.GetType("System.String")));
childDataTable.Columns.Add(new DataColumn("City",
System.Type.GetType("System.String")));
childDataTable.Columns.Add(new DataColumn("StateProvince",
```

```
System.Type.GetType("System.String")));
childDataTable.Columns.Add(new DataColumn("PostalCode",
System.Type.GetType("System.String")));


DataRow childRow = childDataTable.NewRow();

childRow[0] = "241 Ne C St";
childRow[1] = "Willamina";
childRow[2] = "OR";
childRow[3] = "97396-2714";

dataTable.AddChild(row, "Child1", childRow);
dataTable.Rows.Add(row);
```

GetChildren

Retrieves the child rows from a named relationship.

Syntax

```
public EnhancedDataTable GetChildren(DataRow parentRow, string name)
```

Parameters

- ParentRow - the parent row
- Name - the name of the parent/child relationship, e.g. "Flood Plain Data", "References", "Used By", etc.

Results

Returns the child rows from the named relationship.

Example

```
EnhancedDataTable childRows = dataTable.GetChildren(parentRow, "child1");
```

ListChildNames

Retrieves all of the names of the named parent/child relationships.

Syntax

```
public string[] ListChildrenNames(DataRow parentRow)
```

Parameters

None.

Results

Returns the set of the names of the named parent/child relationships.

Example

```
string[] childNames = dataTable.ListChildrenNames( parentRow);
```

SetChildren

Sets the rows of a supplied, named parent/child relationship. If rows previously existed under this name, they will be returned to the caller.

Syntax

```
public void SetChildren(DataRow parentRow, string name, EnhancedDataTable
newTable)
```

Results

Returns the set of the names of the named parent/child relationships.

Example

```
EnhancedDataTable childRows = dataTable1.GetChildren(parentRow, "child1");
dataTable2.SetChildren(otherParentRow, "child1", childRows);
```

10

Using Web Services

In this section:

- Using Web Services 234
- The SOAP Interface 235
- The REST Interface 237

Using Web Services

Introduction to Dynamic Web Services

Services deployed on the Spectrum™ Technology Platform are also published as web services. A web service is a collection of protocols and standards used for exchanging data between applications. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet. Spectrum™ Technology Platform web services are available using the Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) over HTTP.

With this ability, you can:

- Call services via a standard web service interface
- Use a web service interface that is dynamically generated based on the provided metadata
- Pass data into the service and have it returned in the response

Note: This interface is not designed to be a high performance method for accessing the server. If performance is a concern and HTTP is a requirement, the Spectrum™ Technology Platform proprietary XML over HTTP protocol may be the preferred method for calling services from third-party applications.

Accessing Services With a Third-Party Application

This section explains how to use a third-party web service integration tool to generate client code and use the generated code to call the web service. WSDL for a particular service can be obtained by visiting a URL that contains the name of the service appended with ".wsdl". For example, the WSDL for the ValidateAddress service would be the following:

```
http://host:port/services/ValidateAddress?wsdl
```

The following instructions explain, in general, how to access a Spectrum™ Technology Platform service as a web service via a third-party application. However, you should refer to the documentation of your particular tool for exact instructions for that tool.

1. Launch your third-party application.
2. Read the WSDL for the service and generate a client application.
3. Compile the code and invoke the web service. Requests will be input and responses will be returned in document/literal format. A valid Spectrum™ Technology Platform user and password must be provided for the request to pass authentication.

   Note: Alternatively, you can manually write code to access the web service without using a third-party tool.

The SOAP Interface

The Spectrum™ Technology Platform server allows you to access Spectrum™ Technology Platform services using SOAP web services in document/literal mode. This method allows you to create an xml schema to enforce the contract between the client and the server. Document/literal web services are the WS-I compliant format for web services.

Web Services Description Language

Web Services Description Language (WSDL) describes the technical details of how a Web Service works, detailing the technical requirements for uses such as its Internet address and port, method names and associated arguments, data types, and so on.

A WSDL XML document is composed of one or more service descriptions, each in turn composed of a collection of messages, port types, and bindings. A Message is an abstract description of what the exchanged data looks like. A Port Type is a collection of operations, not unlike a programming language library of functions. A Binding associates a Port Type with an appropriate transport protocol, the most common being HTTP.

The WSDL for Spectrum™ Technology Platform web services is automatically created and is accessible via a URL. The URL has the following syntax:

```
http://<host or IP>:<port>/jboss-net/services/<service_name>?wsdl
```

Access to this URL will require admin authentication. Your browser will prompt you for a user name and password. Valid admin-level credentials must be provided to gain access to this web page.

As an example for the ValidateAddress service installed on your local computer, the WSDL would be available at the following URL:

```
http://localhost:port/services/ValidateAddress?wsdl
```

On most platforms a tool will be provided that will consume WSDL and generate proxy classes. Visual Studio .NET provides the Add Web Reference command. Apache Axis provides a WSDL2Java utility. For each service that is to be used, the appropriate proxy classes will need to be generated. Depending on the tool used, the manner in which this operation is performed will vary.

The web service model is a generic model for all services. The WSDL for any Spectrum™ Technology Platform web service will contain the same definition of data types and operations. What differentiates the WSDL is the target service and the values (options and data) that is provided at runtime. The options and values must match what is expected by the target service. Check the service documentation to identify the supported options and data types.

Sample SOAP Request and Response

Following is a sample SOAP request used to call the ValidateAddress component as well as a sample SOAP response. The sections for options and rows are all dependent on the metadata for that particular web service; therefore, different components will have different metadata entries. Additionally, there is a user_fields section that allows you to pass in field values that will be returned, unmodified, in the response.

Note: Since the service name, option name, and field name ultimately become XML elements, they may not contain characters that are invalid in XML element names (for example, spaces are not

valid). Please consult the xml specification for clarification (http://www.w3.org/TR/REC-xml/#NT-Name). Services not meeting this requirement will still function but will not be exposed as a web services.

Sample SOAP Request

Below is a sample SOAP request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:val="http://www.g1.com/services/ValidateAddress">
>
   <soapenv:Header/>
   <soapenv:Body>
      <val:ValidateAddressRequest>
         <val:context>
            <val:account.id>admin</val:account.id>
            <val:account.password>admin</val:account.password>
         </val:context>
         <val:options>
            <val:OutputFormattedOnFail>Y</val:OutputFormattedOnFail>
         </val:options>
         <val:rows>
            <val:row>
               <val:AddressLine1>1525B Kramer Lane</val:AddressLine1>
               <val:AddressLine2>Suite 100</val:AddressLine2>
               <val:AddressLine3></val:AddressLine3>
               <val:AddressLine4></val:AddressLine4>
               <val:City></val:City>
               <val:StateProvince></val:StateProvince>
               <val:PostalCode>78758</val:PostalCode>
               <val:Country></val:Country>
               <val:FirmName></val:FirmName>
               <val:USUrbanName></val:USUrbanName>
               <val:user_fields>
                  <val:user_field>
                     <val:name>myuser</val:name>
                     <val:value>yes</val:value>
                  </val:user_field>
               </val:user_fields>
            </val:row>
         </val:rows>
      </val:ValidateAddressRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample SOAP Response

Below is a sample SOAP response.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ValidateAddressResponse
xmlns="http://www.g1.com/services/ValidateAddress">
         <rows>
            <row>
               <Confidence>88</Confidence>
               <RecordType>Normal</RecordType>
               <CountryLevel>A</CountryLevel>
               <ProcessedBy>USA</ProcessedBy>
```

```
                <MatchScore>0</MatchScore>
                <AddressLine1>1073 Maple Ln</AddressLine1>
                <City>Batavia</City>
                <StateProvince>IL</StateProvince>
                <PostalCode>60510-1135</PostalCode>
                <PostalCode.Base>60510</PostalCode.Base>
                <PostalCode.AddOn>1135</PostalCode.AddOn>
                <Country>United States Of America</Country>
                <user_fields>
      <user_field>
       <name>myuser</val:name>
       <value>yes</val:value>
      </user_field>
     </user_fields>
           </row>
        </rows>
      </ValidateAddressResponse>
   </soap:Body>
</soap:Envelope>
```

The REST Interface

Operations context and parameters as well as response objects are formatted in xml or JavaScript Object Notation (JSON), which is a language-independent, text-based open standard designed for interchange of data structures and objects. The file extension (*.json or *.xml) will determine the format of the response.

Limitations of REST:

- Although REST itself can use multiple HTTP methods, at this time the only method we support is the GET method. As a result, you can pass just one row at a time in a request. Multiple rows and hierarchical data cannot be passed.
- Lists of fields and options must be minimized due to URL length limits, which are set by the browser and web server. We advise you adhere to a limit of 2048 characters, but your browser and web server may allow for more.

The REST service is bound to the http://<host or IP>:<port>/rest/<service_name>results.xml or the http://<host or IP>:<port>/rest/<service_name>results.json path.

Web Application Description Language

Web Application Description Language (WADL) is an XML-based file format that provides a machine-readable description of HTTP-based web applications. These applications are typically REST web services.

WADL is intended for applications that are based on the existing architecture of the Web. Like WSDL, it is platform and language independent and aims to promote re-use of applications beyond the basic use in a web browser.

The service is described using a set of resource elements. Each of these contains param elements to describe the inputs, as well as method elements, which describe the request and response of a resource. The request element specifies how to represent the input, what types are required, and any specific HTTP headers that are required. The response describes the representation of the service's response, as well as any fault information, to deal with errors.

The WADL for Spectrum™ Technology Platform web services is automatically created and is accessible via a URL. The URL has the following syntax:

```
http://<host or IP>:<port>/rest/<service_name>?wadl&type=xml
```

Access to this URL will require admin authentication. Your browser will prompt you for a user name and password. Valid admin-level credentials must be provided to gain access to this web page.

As an example for the ValidateAddress service installed on a local computer, the WADL would be available at the following URL:

```
http://localhost:8080/rest/ValidateAddress?_wadl&_type=xml
```

Sample XML Request and Response

Following is a sample XML request used to call the ValidateAddress component as well as a sample XML response. The sections for options and rows are all dependent on the metadata for that particular web service; therefore, different components will have different metadata entries.

Additionally, you can pass extra fields through the service even if the service doesn't use them. These fields are passed along with the fields the service *does* use, and are returned, unmodified, in the user_fields section of the response. For example, if you wanted to pass a field called "customID", the following text would be added to the request URL: "Data.customID=xxx".

Note: Since the service name, option name, and field name ultimately become XML elements, they may not contain characters that are invalid in XML element names (for example, spaces are not valid). Please consult the xml specification for clarification (http://www.w3.org/TR/REC-xml/#NT-Name). Services not meeting this requirement will still function but will not be exposed as a web services.

Sample XML Request

Below is a sample XML request.

```
http://localhost:8080/rest/ValidateAddress/results.xml+Option.Database.US=USA&
Data.AddressLine1=1825%20Kramer%20Lane&Data.PostalCode=78759
```

Sample XML Response

Below is a sample XML response.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <xml.ValidateAddressResponse
xmlns="http://www.pb.com/spectrum/services/ValidateAddress">
    <output_port>
    <Address>
    <Confidence>82</Confidence>
    <RecordType>Normal</RecordType>
    <CountryLevel>A</CountryLevel>
    <ProcessedBy>USA</ProcessedBy>
    <MatchScore>0</MatchScore>
    <AddressLine1>1825 Kramer Ln</AddressLine1>
    <City>Austin</City>
    <StateProvince>TX</StateProvince>
    <PostalCode>78758-4260</PostalCode>
    <PostalCode.Base>78758</PostalCode.Base>
    <PostalCode.AddOn>4260</PostalCode.AddOn>
```

```
    <Country>United States Of America</Country>
    <user_fields/>
    </Address>
    </output_port>
    </xml.ValidateAddressResponse>
```

Part III: Modules

In this section:

- Address Now Module 243
- Enterprise Geocoding Module 293
- Enterprise Routing Module 613
- Enterprise Tax Module 647
- GeoConfidence Module 701
- Location Intelligence Module 705
- Universal Addressing Module 721

11

Address Now Module

In this section:

- What Is the Address Now Module? 244
- BuildGlobalAddress 245
- GetGlobalCandidateAddresses 257
- ValidateGlobalAddress 264
- The ACR Code 289

What Is the Address Now Module?

The Address Now Module is an address standardization and validation tool that provides comprehensive coverage for addresses outside the U.S. and Canada. Address Now is one of two address standardization and validation modules available for Spectrum™ Technology Platform. The other module is the Universal Addressing Module. The Address Now Module provides the following benefits over the Universal Addressing Module for addresses outside the U.S. and Canada:

- **Better data**—The database used by the Address Now Module is more up-to-date and complete in many countries than the database used by the Universal Addressing Module. This is because the Universal Addressing Module relies on data from Universal Postal Union (UPU), a body of the United Nations, for its international data and while the data coverage is extensive, the updates and the level of address details are not proactively managed by the UPU. Address Now on the other hand relies on data directly from the postal authorities (in most countries), plus other third party data providers. This means that the data is more current with postal changes and is more detailed.
- **Drill down feature**—The Address Now Module also offers drill down capabilities to address data from any country allowing users to rapidly enter address information without having to worry about the structure or making data entry mistakes.
- **Double-byte support**—The Address Now Module is Unicode enabled, recognizing kanji and other double-byte characters.

Address Now Components

Address Now consists of the following components. These components can work with U.S., Canadian, and international addresses.

- **BuildGlobalAddresses**—Allows you to interactively build an address by searching for individual address elements.
- **GetGlobalCandidateAddresses**—Returns a list of addresses that are considered matches for a given address.
- **ValidateGlobalAddress**—Standardizes addresses using international postal data. ValidateGlobalAddress can also validate addresses in the U.S. and Canada but its strength is validation of addresses in other countries. If you have a significant amount of non-U.S. and non-Canadian address data, you should consider using ValidateGlobalAddress.

In cases where ValidateGlobalAddress returns multiple address matches for a given input address, you can use GetGlobalCandidateAddresses to return the address stack. GetGlobalCandidateAddresses is designed to return additional information from the postal databases to help you determine which of the returned addresses is the best match.

Address Now Database

The Address Now database contains postal data from all supported countries. You can install the entire database or the data for specific countries only. The database is installed on the server. This database is available by subscription from Pitney Bowes Software and is updated monthly.

BuildGlobalAddress

BuildGlobalAddress allows you to build a valid address starting with just a single address element or a few address elements.

BuildGlobalAddress is part of the Address Now Module. For more information on the Address Now Module, including a listing of other components included with Address Now, see **What Is the Address Now Module?** on page 244.

Using BuildGlobalAddress

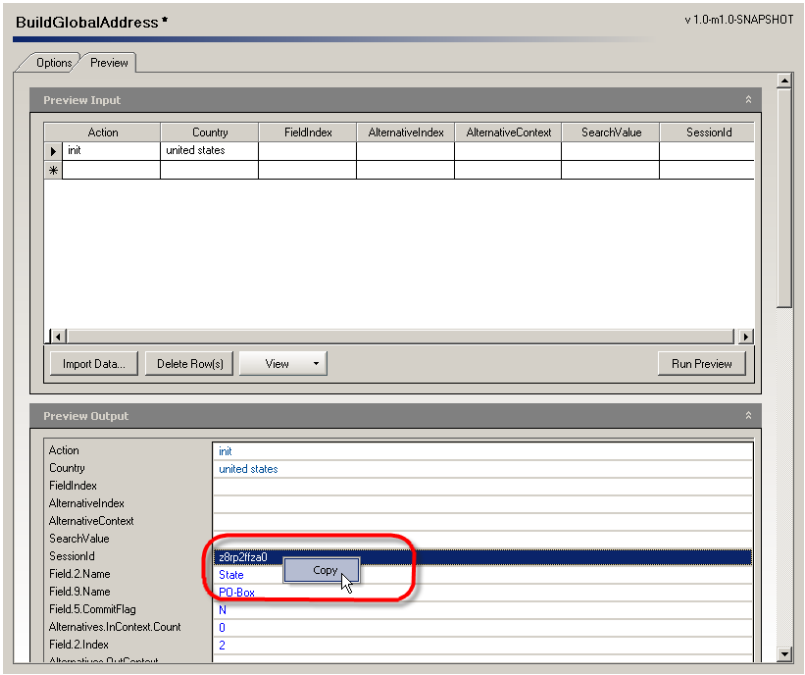
BuildGlobalAddress works differently than other Spectrum™ Technology Platform components. Building an address is an interactive process that requires you to select address elements at each step of the address building process. This means that building an address requires a sequence of calls to BuildGlobalAddress, not a single call. To start, you make an initialization call to BuildGlobalAddress. This call returns a session ID. You then use this session ID in subsequent calls. With each call, BuildGlobalAddress presents a list of alternative values for an address element. You select the value you want, then move on to the next address element until the complete address is built. With some exceptions, you need to make a separate call for each address element.

The overall process works like this:

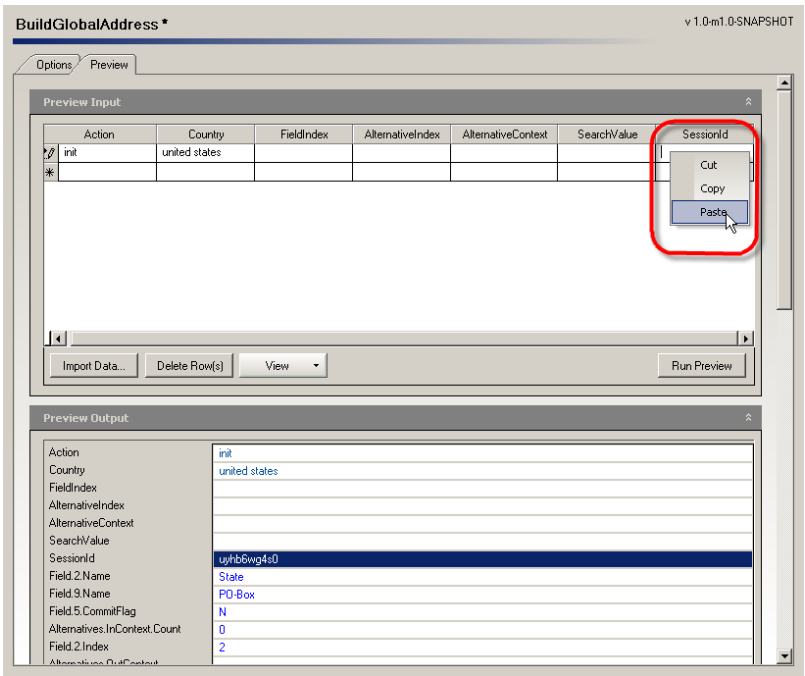
- First, you make an initialization call to open a session and receive a system-assigned session ID.
- Make a search call to find possible values for a given address element.
- When you have selected the value you want, you make a commit call to indicate the value you want for the given address element.
- Continue to make search/commit calls until all address elements are committed.
- Finally, you make a close call to end the session.

To familiarize yourself with how the process works, use the Management Console's Preview tab to step through the following procedure.

1. Open the Management Console.
2. Under the Services node, select `Build Global Address`.
3. On the **Options** tab, specify the options you want. For information on the options, see **Options** on page 250.
4. Click the **Preview** tab.
5. In the **Action** field type `init`.
6. In the **Country** field enter the country of the address you want to build.
7. Click **Run Preview**.
8. Under Preview Output, find the **SessionId** field, right-click the value, and select `Copy` from the pop-up menu.



9. Under Preview Input, right-click the **SessionId** field and select `Paste`.



10. Enter the following values in the input fields:

- Action—Type search.
- Country—Keep this field the same.
- FieldIndex—Type the index value of the first field you want to search. For example, if you know you want to search for an address in Chicago, you would type "1" because for U.S. addresses, field index 1 corresponds to the City field.
- SearchValue—Type the value you want to search for. For example, if you want to build an address in Chicago, you would type "chicago".
- SessionId—Keep the same value.

Note: The values in the other input fields are ignored.

11. Click **Run Preview** again.
12. The results of the search are placed in up to two output fields: **Alternatives.InContext** and **AlternativesOutContext**. For an explanation of the difference between in context results and out of context results, see **What Is Context?** on page 256.
13. When you have found the value you want, enter the following values in the input fields:

    - **Action**—Type **commit**.
    - **AlternativeIndex**—Type the index number for the alternative you choose. Index values start with 0, not 1. For example, if you search for Chicago the alternatives returned by BuildGlobalAddress would be indexed as follows. If you want to commit the value "CHICAGO" you would type "0" in the AlternativeIndex field.

      - 0—CHICAGO
      - 1—CHICAGO HTS
      - 2—CHICAGO PARK
      - 3—CHICAGO RIDGE
      - 4—EAST CHICAGO
      - 5—NORTH CHICAGO
      - 6—WEST CHICAGO

    - **AlternativeContext**—Type `in` or `out` to indicate whether the index value you specified in **AlternativeIndex** is for the list of alternatives in the **Alternatives.InContext** field or the **Alternatives.OutContext** field.
    - **SessionId**—Keep this value the same.

    Note: The values in the other input fields are ignored.

14. Click **Run Preview** again. The value you specified will now be in the Field.n.Value field for the appropriate address element.
15. Repeat the search and commit steps as often as needed until you have built the address.
16. Close the session by entering the following values in the input fields:

    - **Action**—Type **close**.
    - **SessionId**—Keep this value the same.

    Note: The values in the other input fields are ignored.

Input

Note: specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 51: BuildGlobalAddress Input

columnName	Format	Description / Valid Values
Action	String	Specifies the action to take. One of the following: **init** Initialization. This action opens a session and returns a session ID which is required for all other actions. The init action requires the Country input field. **search** Searches for values for a specific address element and returns a list of alternative values for you to choose. The search action requires the following input fields: • FieldIndex • SearchValue • SessionId **commit** Assigns one of the values returned by the search action to the field. The commit action requires the following input fields: • AlternativeIndex • AlternativeContext • SessionId **clear** Un-commits the field specified in the FieldIndex field. The clear action requires the following input fields: • FieldIndex • SessionID **close** Ends a session. The close action requires the SessionId input field.
AlternativeContext	String	For the commit action, indicates whether you are choosing a value from the **Alternatives.InContext** field or the

columnName	Format	Description / Valid Values
		Alternatives.OutContext field. This field is ignored for other actions. One of the following: **in** You are committing a value from the **Alternatives.InContext** field. This means that the value you specify in the **AlternativeIndex** input field corresponds to a value in the **Alternatives.InContext** output field. **out** You are committing a value from the **Alternatives.OutContext** field. This means that the value you specify in the **AlternativeIndex** input field corresponds to a value in the **Alternatives.OutContext** output field.
AlternativeIndex	String [79]	For the commit action, specifies the value you want to use in the address you are building. For example, if you searched for a city and BuildGlobalAddress returns a list of three cities, you would indicate the city you want by specifying the index value for your choice. Index values for the alternatives presented by BuildGlobalAddress are zero-based, meaning that the first alternative has an index of 0, the second alternative has a value of 1, and so on. The input field is ignored for actions other than commit.
Country	String [79]	For the init action, specifies the country in which you want to build an address. Specify the country using the format you chose for input country format (English name two-character ISO 3116-1 Alpha-2 code, or three-character ISO 3116-1 Alpha-3 code). For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908. This input field is ignored for actions other than init.
FieldIndex	String [79]	For the search action, specifies the address element that you want to search on. For the clear action, specifies the address element you want to un-commit. One of the following: **all** performs the "clear" action on all address elements. This option applies to the "clear" action only. **<IndexNumber>** Performs the action on a specific address element. To determine the index of an address element, first look at the Field.n.Name fields and locate the field

columnName	Format	Description / Valid Values
		you want. The value n indicates the field's index. For example, you want to look up ZIP Codes for U.S. addresses. After the init call you see that Field.0.Name is "Zip" indicating that the ZIP Code has a field index of "0". This input field is ignored for actions other than search and clear.
SearchValue	String [79]	For the search action, specifies the value you want to search for. This value must be appropriate for the field you specified in FieldIndex. For example, if you specified the ZIP Code field in FieldIndex, then you would enter a ZIP Code or partial ZIP Code in this field. Likewise if you chose the city field in FieldIndex you would specify a city name or partial city name in this field. if you leave the field blank the search will return all values that are in context. For more information about in context and out of context values, see **What Is Context?** on page 256. This input field is ignored for actions other than search.
SessionId	String [79]	Specifies the session ID you want to use for this call. To obtain a session ID use the init action. If a session is inactive for 5 minutes it will expire and you will need to perform a new init call to start a new session. This field is required for all actions except init.

Options

Table 52: BuildGlobalAddress Options

optionName	Description/Valid Values
HomeCountry	Specifies the default country. You should specify the country where most of the addresses in your data are located. For example, if most of your addresses are in Canada, specify Canada. BuildGlobalAddress uses the country you specify to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields.

optionName	Description/Valid Values
OutputCountryFormat	Specifies the format to use for the country name in the output. One of the following: **E** The country in the output in English (default). **I** The country in the output as the two-character ISO code. **U** The country in the output as the three-character UPU code.
OutputPostalCodeSeparator	Specifies whether or not to use separators (spaces or hyphens) in ZIP Codes or Canadian postal codes. For example, a ZIP + 4® Code with the separator would be 20706-1844 and without the separator it would be 207061844. A Canadian postal code with the separator would be P5E"1S7 and without the separator it would be P5E1S7. **Y** Yes, use separator (default). **N** No, do not use separator. **Note:** Spaces are used in Canadian postal codes and hyphens in U.S. ZIP + 4® Codes.
ShowExtraAddressLine	Specifies whether or not to include the city, state/province, and postal code in one of the AddressLine output fields. Regardless of what you specify with this option, the output fields City, State/Province, and PostalCode will always contain the city, state/province, and postal code. **Y** Yes, include city, state/province, and postal code in an AddressLine output field (default). **N** No, do not include city, state/province, and postal code in an AddressLine output field.
MaximumResults	Allows you to set the default value for this option to any value from 1 to 10000; it has a default value of 50 records. Note that values set in Enterprise Designer override those set in Management Console

Output

BuildGlobalAddress returns address data and return codes for each input address.

Note: If you are using the API to interact with this component, the output is returned in the DataTable class. The fields described below are the valid column names in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section of the *API User's Guide*.

Address Data

Table 53: BuildGlobalAddress Output

columnName	Format	Description / Valid Values
Action	String [79]	Shows the value specified in the Action input field for this call. For more information on this input field see **Input** on page 248.
AddressLine1	String [79]	The formatted first address line.
AddressLine2	String [79]	The formatted second address line.
AddressLine3	String [79]	The formatted third address line.
AddressLine4	String [79]	The formatted fourth address line.
AddressLine5	String [79]	The formatted fifth address line.
AddressLine6	String [79]	The formatted sixth address line.
AddressLine7	String [79]	The formatted seventh address line.
AddressLine8	String [79]	The formatted eighth address line.
AlternativeContext	String [79]	Shows the value specified in the AlternativeContext input field for this call. For more information, see **Input** on page 248.
AlternativeIndex	String [79]	Shows the value specified in the AlternativeIndex input field for this call. For more information on this input field see **Input** on page 248.
Alternatives.InContext	String [79]	A comma-delimited list of the possible values for the field you searched on which fit the context of fields you have already committed. For information on context see **What Is Context?** on page 256.
Alternatives.InContext.Count	String [79]	The number of "in context" results returned by your search. For information on context see **What Is Context?** on page 256.
Alternatives.OutContext	String [79]	A comma-delimited list of the possible values for the field you searched on which do not fit the context of fields you have already committed. For

columnName	Format	Description / Valid Values
		information on context see **What Is Context?** on page 256.
Alternatives.OutContext.Count	String [79]	The number of "out of context" results returned by your search. For information about context, see **What Is Context?** on page 256.
ApartmentLabel	String [79]	Apartment designator (such as STE or APT). For example: 123 E Main St. APT 3
ApartmentNumber	String [79]	Apartment number. For example: 123 E Main St. APT **3**
Building	String [79]	The name of a building.
City	String [79]	The city name.
Country	String [79]	Shows the value specified in the Country input field for this call. For more information about this input field, see **Input** on page 248.
Country	String [79]	The two- or three-character ISO code, or English name of the country. For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908.
Department	String [79]	The name of a distinct part of anything arranged into divisions. For example, the Finance Department in a corporation.
Field.n.CommitFlag	String [79]	Indicates whether you have chosen a value for field n (i.e. "committed" a value). One of the following: **Y** Yes, the value of this field has been committed. **N** No, the value of this field has not been committed.
Field.n.Index	String [79]	An index value used to refer to field n, where n is 0 though 10. For example, for U.S. addresses the index value of the ZIP field is "0".

columnName	Format	Description / Valid Values
Field.n.Name	String [79]	The name of the address element contained in field n, where n is 0 through 10. For example, for U.S. addresses Field.0.Name is ZIP.
Field.n.Value	String [79]	The value that has been committed to field n, where n is 0 through 10. This field is blank on the init call.
FieldIndex	String [79]	Shows the value specified in the FieldIndex input field for this call. For more information on this input field see **Input** on page 248.
FirmName	String [79]	The name of a company. For example: **Pitney Bowes Software** 4200 PARLIAMENT PL STE 600 LANHAM MD 20706-1844 USA
HouseNumber	String [79]	House number. For example: **123** E Main St. Apt 3
POBox	String [79]	The post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode	String [79]	The postal code. In the U.S. this is the ZIP Code™.
PostalCode.AddOn	String [79]	The 4-digit add-on part of the ZIP + 4® Code. For example, in the ZIP Code™ 60655-1844, 1844 is the 4-digit add-on. (U.S. addresses only.)
PostalCode.Base	String [79]	The 5-digit ZIP Code™. For example 20706 (U.S. addresses only.)
Principality	String [79]	An area within a country. For example, England, Scotland, and Wales are principalities. This field will normally be blank.
SearchFieldIndex	String [79]	The index value of the field searched in the previous search action.
SearchValue	String [79]	Shows the value specified in the SearchValue input field for this call. For more information on this input field see **Input** on page 248.

columnName	Format	Description / Valid Values
SessionId	String [79]	Shows the value specified in the SessionId input field for this call. For more information on this input field see **Input** on page 248.
StateProvince	String [79]	The state or province abbreviation.
StreetName	String [79]	Street name. For example: 123 E **Main** St. Apt 3
StreetSuffix	String [79]	Street suffix. For example: 123 E Main **St**. Apt 3
SubCity	String [79]	A district or suburb. The subcity is used in countries where it is common to include the district or suburb within the address. For example, 27 Crystal Way **Bradley Stoke** Bristol BS32 8GA In this case, "Bradley Stoke" is the subcity.
SubStreet	String [79]	The second street address used to identify an address. Substreets are used in countries where it is common to give two street names in the address. For example, 12 The Mews **High Street** In this example, "High Street" is the substreet. Substreets can be used to precisely identify the delivery location. In the example, "The Mews" may be a small street that needs another street identification to properly locate the address, so "High Street" is included. In this case, "High Street" is the main or known street.
USCountyName	String [79]	For U.S. addresses, the name of the county where the address is located.

Return Codes

Table 54: BuildGlobalAddress Return Codes

columnName	Format	Description / Valid Values
Status	String [79]	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	String [79]	Reason for failure, if there is one. • SessionError • SeverError • CountryNotFound
Status.Description	String [79]	Description of the problem, if there is one. **Please initialize new session** This value will appear if Status.Code=SessionError. **Null or empty action** This value will appear if Status.Code=SessionError. **Unknown action** This value will appear if Status.Code=SessionError. **Invalid session** This value will appear if Status.Code=SessionError. **Invalid value for** This value will appear if Status.Code=SessionError. **Cannot Search Committed Field** This value will appear if Status.Code=SessionError. **Module not licensed** This value will appear if Status.Code=ServerError. **Could Not Identify Country** This value will appear if Status.Code=CountryNotFound

What Is Context?

When you perform a search for an address element, BuildGlobalAddress looks at the address elements that you have already committed and splits up the values it returns based on whether or not the returned values exist within the context of the address elements you have already committed. For example, in the U.S. the following cities exist:

In Illinois:

- CHICAGO
- CHICAGO HTS
- CHICAGO RIDGE
- NORTH CHICAGO
- WEST CHICAGO

In Indiana:

- EAST CHICAGO

In Nevada:

- CHICAGO PARK

If you have already committed a value of "IN" (Indiana) for the state and then searched for the city "chicago", BuildGlobalAddress would return EAST CHICAGO as an "in context" result because it exists in Indiana, and it would return all the other matches for "chicago" as out-of-context results. Likewise, if you committed a value of "IL" (Illinois) for the state, BuildGlobalAddress would return EAST CHICAGO and CHICAGO PARK as out of context, and CHICAGO, CHICAGO HTS, CHICAGO RIDGE, NORTH CHICAGO, and WEST CHICAGO as "in context."

GetGlobalCandidateAddresses

GetGlobalCandidateAddresses returns a list of addresses that are considered matches for a given input address. If the input address matches multiple addresses in the Address Now database, the possible matches are returned. If the input address matches only one address in the Address Now database, no address data is returned.

GetGlobalCandidateAddresses is part of the Address Now Module. For more information on the Address Now Module, including a listing of other components included with the Address Now Module, see **What Is the Address Now Module?** on page 244.

Input

GetGlobalCandidateAddresses takes a standard address as input. All addresses use this format no matter what country the address is from. AddressLine1 and Country are required input fields. The other fields are optional.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 55: GetGlobalCandidateAddresses Input

columnName	Format	Description / Valid Values
AddressLine1	String [79]	First address line. This is a required field.
AddressLine2	String [79]	Second address line
AddressLine3	String [79]	Third address line
AddressLine4	String [79]	Fourth address line
AddressLine5	String [79]	Fifth address line
AddressLine6	String [79]	Sixth address line
AddressLine7	String [79]	Seventh address line
AddressLine8	String [79]	Eighth address line
City	String [79]	City name
StateProvince	String [79]	State or province.
PostalCode	String [10]	The postal code for the address in one of these formats: 99999 99999-9999 A9A9A9 A9A 9A9 9999 999

columnName	Format	Description / Valid Values
Country	String	The country. Specify the country using the format you chose for input country format (English name or ISO code). For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	String [79]	Company or firm name

Options

Table 56: GetGlobalCandidateAddresses Options

optionName	Description/Valid Values
HomeCountry	Specifies the default country. Specify the country that is the destination of most of your mailpieces. For example, if most of your mailpieces are going to Canada, specify Canada. GetGlobalCandidateAddresses uses the country you specify to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields. For a list of valid country names, see **Country ISO Codes and Module Support** on page 908.
OutputCountryFormat	Specifies the format to use for the country name in the output. One of the following: **E** The country in the output in English (default). **I** The country in the output in the two-character ISO code. **U** The country in the output in the three-character UPU code.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputPostalCodeSeparator	Specifies whether or not to use separators (spaces or hyphens) in ZIP™ Codes or Canadian postal codes. For example, a ZIP + 4® Code with the separator would be 20706-1844 and without the separator it would be 207061844. A Canadian postal code with the separator would be P5E"1S7 and without the separator it would be P5E1S7.

optionName	Description/Valid Values
	Y Yes, use separator (default) **N** No, do not use separator **Note:** Spaces are used in Canadian postal codes and hyphens in U.S. ZIP + 4® Codes.
ShowExtraAddressLine	Specifies whether or not to include the city, state/province, and postal code in one of the AddressLine output fields. Regardless of what you specify with this option, the output fields City, State/Province, and PostalCode will always contain the city, state/province, and postal code. **Y** Yes, include city, state/province, and postal code in an AddressLine output field (default). **N** No, do not include city, state/province, and postal code in an AddressLine output field.
MaximumResults	The maximum number of candidate addresses to output. The default is 50. The maximum value is 100.
ReturnUserData	Specifies whether or not to include in the output data from the input address that could not be validated. **Y** Yes, include input data that could not be validated. **N** No, do not include input data that could not be validated (default).

Output

GetGlobalCandidateAddresses returns address data and return codes for each address.

Note: he output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Address Data

Table 57: GetGlobalCandidateAddresses Address Data Output

columnName	Format	Description / Valid Values
AddressLine1	String [79]	The formatted first address line.

columnName	Format	Description / Valid Values
AddressLine2	String [79]	The formatted second address line.
AddressLine3	String [79]	The formatted third address line.
AddressLine4	String [79]	The formatted fourth address line.
AddressLine5	String [79]	The formatted fifth address line.
AddressLine6	String [79]	The formatted sixth address line.
AddressLine7	String [79]	The formatted seventh address line.
AddressLine8	String [79]	The formatted eighth address line.
ApartmentLabel	String [79]	Apartment designator (such as STE or APT). For example: 123 E Main St. **APT** 3
ApartmentNumber	String [79]	Apartment number. For example: 123 E Main St. APT **3**
Building	String [79]	The name of a building.
City	String [79]	The city name.
Country	String [79]	The ISO code or English name of the country. For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908.
Department	String [79]	The name of a distinct part of anything arranged into divisions. For example, the Finance Department in a corporation.
FirmName	String [79]	The name of a company. For example: **Pitney Bowes Software** 4200 PARLIAMENT PL STE 600

columnName	Format	Description / Valid Values
		LANHAM MD 20706-1844 USA
HouseNumber	String [79]	House number. For example: **123** E Main St. Apt 3
POBox	String [79]	Post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode	String [79]	The postal code as required by the local postal authority. For example, in the U.S. the postal code is the ZIP Code.
PostalCode.AddOn	String [79]	For U.S. addresses, the last four digits of the ZIP + 4® Code.
PostalCode.Base	String [79]	For U.S. addresses, the five-digit ZIP Code.
Principality	String [79]	An area within a country. For example, England, Scotland, and Wales are principalities. This field will normally be blank.
StateProvince	String [79]	The state or province abbreviation.
StreetName	String [79]	Street name. For example: 123 E **Main** St. Apt 3
StreetSuffix	String [79]	Street suffix. For example: 123 E Main **St**. Apt 3
SubCity	String [79]	A district or suburb. The subcity is used in countries where it is common to include the district or suburb within the address. For example, 27 Crystal Way **Bradley Stoke** Bristol BS32 8GA In this case, "Bradley Stoke" is the subcity.
SubStreet	String [79]	The second street address used to identify an address. Substreets are used in countries where it is common to give two street names in the address. For example, 12 The Mews **High Street**

columnName	Format	Description / Valid Values
		In this example, "High Street" is the substreet. Substreets can be used to precisely identify the delivery location. In the example, "The Mews" may be a small street that needs another street identification to properly locate the address, so "High Street" is included. In this case, "High Street" is the main or known street.
USCountyName	String [79]	For U.S. addresses, the name of the county where the address is located.

Return Codes

Table 58: GetGlobalCandidateAddresses Return Codes

columnName	Format	Description / Valid Values
ACRCode	String [79]	The Address Correction Result (ACR) code describes what data has been changed in each record. For information on what this code means, see **The ACR Code** on page 289.
Confidence	String [79]	The level of confidence assigned to the address being returned. Range is from zero (0) to 100; zero indicates failure, 100 indicates a very high level of confidence that the match results are correct.
Status	String [79]	Reports the success or failure of the match attempt. **Null** Success **F** Failure
Status.Code	String [79]	Reason for failure, if there is one. • RequestFailed • ServerError • CountryNotFound
Status.Description	String [79]	Description of the problem, if there is one. **Maximum records cannot be set to 0. Minimum value should be 1** This value will appear if Status.Code=RequestFailed. **Address Not Found** This value will appear if Status.Code=RequestFailed.

columnName	Format	Description / Valid Values	
		Module not licensed	This value will appear if Status.Code=ServerError.
		Could Not Identify Country	This value will appear if Status.Code=CountryNotFound.

ValidateGlobalAddress

ValidateGlobalAddress provides enhanced address standardization and validation for addresses outside the U.S. and Canada. ValidateGlobalAddress can also validate addresses in the U.S. and Canada but its strength is validation of addresses in other countries. If you send a significant amount of mail to destinations outside the U.S. and Canada, you should consider using ValidateGlobalAddress.

ValidateGlobalAddress is part of the Address Now Module. For more information on the Address Now Module, including a listing of other components included with the Address Now Module, see **What Is the Address Now Module?** on page 244.

Input

ValidateGlobalAddress takes a standard address as input. All addresses use this format no matter what country the address is from.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 59: ValidateGlobalAddress Input

columnName	Format	Description / Valid Values
AddressLine1	String [79]	First address line
AddressLine2	String [79]	Second address line
AddressLine3	String [79]	Third address line
AddressLine4	String [79]	Fourth address line

columnName	Format	Description / Valid Values
AddressLine5	String [79]	Fifth address line
AddressLine6	String [79]	Sixth address line
AddressLine7	String [79]	Seventh address line
AddressLine8	String [79]	Eighth address line
City	String [79]	City name
StateProvince	String [79]	State or province.
PostalCode	String [79]: 99999 ~~999999999~~ A9A9A9 A9A 9A9 9999 999	The postal code for the address. In the U.S. this is the ZIP Code™.
Country	String [79]	Specify the country using the format you chose for input country format (English name or ISO code). For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	String [79]	Company or firm name

Options

Input Data Options

Table 60: ValidateGlobalAddress Input Data Options

optionName	Description/Valid Values
HomeCountry	Specifies the default country. You should specify the country where most of the addresses are located. For example, if most of the addresses you process are in Canada, specify Canada. ValidateGlobalAddress uses the home country to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields. For a list of valid values, see **Country ISO Codes and Module Support** on page 908.

Output Data Options

Table 61: ValidateGlobalAddress Output Data Options

optionName	Description/Valid Values
OutputCountryFormat	Specifies the format to use for the country name in the output. One of the following: **E** The country in the output is English (default). **I** The country in the output in the two-character ISO code. **U** The country in the output in the three-character UPU code.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputPostalCodeSeparator	Specifies whether to use a separator (spaces or hyphens) in ZIP™ Codes or Canadian postal codes. For example, a ZIP + 4® Code with the separator would be 20706-1844 and without the separator it would be 207061844. A Canadian postal code with the separator would be P5E"1S7 and without the separator it would be P5E1S7. **Y** Yes, use separator (default)

optionName	Description/Valid Values
	N No, do not use separator **Note:** Spaces are used in Canadian postal codes and hyphens in U.S. ZIP + 4® Codes.
ShowExtraAddressLine	Specifies whether to include the city, state/province, and postal code in one of the AddressLine output fields. Regardless of what you specify with this option, the output fields City, State/Province, and PostalCode will always contain the city, state/province, and postal code. **Y** Yes, include city, state/province, and postal code in an AddressLine output field (default). **N** No, do not include city, state/province, and postal code in an AddressLine output field.
StandardizeAddressOnFail	Specifies whether to return a standardized address when an address cannot be validated. The address is formatted using the preferred address format for the address's country. If this option is not selected, the output address component fields (StreetName, HouseNumber, etc.) are blank when address validation fails. **N** No, do not format failed addresses (default). **Y** Yes, standardize failed addresses.
FormatOnFail	Specifies whether to return a formatted address when an address cannot be validated. The address is formatted using the preferred address format for the address's country. One of the following: **Y** Yes, return a formatted address when an address cannot be validated. **N** No, do not return a formatted address when an address cannot be validated (default).
ValidateAddress	Enables address validation. Address validation does the following: • Matches components to the relevant country's reference data • Corrects spelling errors • Adds missing components • Corrects or adds postal codes One of the following: **Y** Yes, validate addresses (default). **N** No, do not validate addresses.

optionName	Description/Valid Values
FormatAddress	Formats the address components into the statutory postal or custom formats. One of the following: **Y** Yes, format addresses (default). **N** No, do not format addresses.

Standardization Options

Table 62: ValidateGlobalAddress Standardization Options

Option optionName	Description/Valid Values
StandardizeComponent.Department	Specifies whether or not to populate the Department field when standardizing an address. **Y** Yes (default) **N** No
StandardizeComponent.FirmName	Specifies whether or not to populate the FirmName field when standardizing an address. **Y** Yes (default) **N** No
StandardizeComponent.Building	Specifies whether or not to populate the Building field when standardizing an address. **Y** Yes (default) **N** No
StandardizeComponent.SubBuilding	Specifies whether or not to populate the SubBuilding field when standardizing an address. **Y** Yes (default) **N** No
StandardizeComponent.HouseNumber	Specifies whether or not to populate the HouseNumber field when standardizing an address.

Option optionName	Description/Valid Values	
	Y	Yes (default)
	N	No
StandardizeComponent.SubStreet	Specifies whether or not to populate the SubStreet field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.StreetName	Specifies whether or not to populate the StreetName field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.POBox	Specifies whether or not to populate the POBox field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.SubCity	Specifies whether or not to populate the SubCity field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.City	Specifies whether or not to populate the City field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.USCountyName	Specifies whether or not to populate the USCountyName field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.StateProvince	Specifies whether or not to populate the StateProvince field when standardizing an address.	

Option optionName	Description/Valid Values	
	Y	Yes (default)
	N	No
StandardizeComponent.Principality	Specifies whether or not to populate the Principality field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.PostalCode	Specifies whether or not to populate the PostalCode field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.Plus4	Specifies whether or not to populate the +4 field when standardizing an address.	
	Y	Yes (default)
	N	No
StandardizeComponent.Country	Specifies whether or not to populate the Country field when standardizing an address.	
	Y	Yes (default)
	N	No
ReportVulgarWords	Specifies whether or not to look for vulgar words. If this option is enabled, ValidateGlobalAddress returns a value in the WCRCode output field to indicate the results.	
	Y	Yes
	N	No (default)
FlagVulgarWords	Specifies whether or not to mark vulgar words in the output using the format ">VulgarWord<".	
	Y	Yes
	N	No (default)

Option optionName	Description/Valid Values
DebugOutput	This option controls whether or not to include troubleshooting information in the output fields Email1, Email2, URL1, and URL2.One of the following: **Y** Yes **N** No (default)

Validation Options

Table 63: ValidateGlobalAddress Validation Options

optionName	Description/Valid Values
ValidateComponent.Department	Specifies whether or not to include the Department field when validating an address. **Y** Yes **N** No (default)
ValidateComponent.FirmName	Specifies whether or not to include the FirmName field when validating an address. **Y** Yes **N** No (default)
ValidateComponent.Building	Specifies whether or not to include the Building field when validating an address. **Y** Yes (default) **N** No
ValidateComponent.SubBuilding	Specifies whether or not to include the SubBuilding field when validating an address. **Y** Yes (default) **N** No

optionName	Description/Valid Values	
ValidateComponent.HouseNumber	Specifies whether or not to include the HouseNumber field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.SubStreet	Specifies whether or not to include the SubStreet field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.StreetName	Specifies whether or not to include the StreetName field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.POBox	Specifies whether or not to include the POBox field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.SubCity	Specifies whether or not to include the SubCity field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.City	Specifies whether or not to include the City field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.USCountyName	Specifies whether or not to include the USCountyName field when validating an address.	
	Y	Yes (default)

optionName	Description/Valid Values	
	N	No
ValidateComponent.StateProvince	Specifies whether or not to include the StateProvince field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.Principality	Specifies whether or not to include the Principality field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.PostalCode	Specifies whether or not to include the PostalCode field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.Plus4	Specifies whether or not to include the +4 field when validating an address.	
	Y	Yes (default)
	N	No
ValidateComponent.Country	Specifies whether or not to include the Country field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.Department	Specifies whether or not to correct the Country field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.FirmName	Specifies whether or not to correct the FirmName field when validating an address.	

optionName	Description/Valid Values	
	Y	Yes (default)
	N	No
ForceUpdate.Building	Specifies whether or not to correct the Building field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.SubBuilding	Specifies whether or not to correct the SubBuilding field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.HouseNumber	Specifies whether or not to correct the HouseNumber field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.SubStreet	Specifies whether or not to correct the SubStreet field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.StreetName	Specifies whether or not to correct the StreetName field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.POBox	Specifies whether or not to correct the POBox field when validating an address.	
	Y	Yes (default)
	N	No

optionName	Description/Valid Values	
ForceUpdate.SubCity	Specifies whether or not to correct the SubCity field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.City	Specifies whether or not to correct the City field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.USCountyName	Specifies whether or not to correct the USCountyName field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.StateProvince	Specifies whether or not to correct the StateProvince field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.Principality	Specifies whether or not to correct the Principality field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.PostalCode	Specifies whether or not to correct the PostalCode field when validating an address.	
	Y	Yes (default)
	N	No
ForceUpdate.Plus4	Specifies whether or not to correct the +4 field when validating an address.	
	Y	Yes (default)

optionName	Description/Valid Values	
	N	No
ForceUpdate.Country	Specifies whether or not to correct the Country field when validating an address.	
	Y	Yes (default)
	N	No
ReplaceAlias.Department	Specifies whether or not to overwrite the Department field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.FirmName	Specifies whether or not to overwrite the FirmName field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.Building	Specifies whether or not to overwrite the Building field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.SubBuilding	Specifies whether or not to overwrite the SubBuilding field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.HouseNumber	Specifies whether or not to overwrite the HouseNumber field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.SubStreet	Specifies whether or not to overwrite the SubStreet field if an alias is found in the Address Now database.	

optionName	Description/Valid Values	
	Y	Yes
	N	No (default)
ReplaceAlias.StreetName	Specifies whether or not to overwrite the StreetName field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.POBox	Specifies whether or not to overwrite the POBox field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.SubCity	Specifies whether or not to overwrite the Subcity field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.City	Specifies whether or not to overwrite the City field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.USCountyName	Specifies whether or not to overwrite the USCountyName field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)
ReplaceAlias.StateProvince	Specifies whether or not to overwrite the StateProvince field if an alias is found in the Address Now database.	
	Y	Yes
	N	No (default)

optionName	Description/Valid Values
ReplaceAlias.Principality	Specifies whether or not to overwrite the Principality field if an alias is found in the Address Now database. **Y** Yes **N** No (default)
ReplaceAlias.PostalCode	Specifies whether or not to overwrite the PostalCode field if an alias is found in the Address Now database. **Y** Yes (default) **N** No
ReplaceAlias.Plus4	Specifies whether or not to overwrite the +4 field if an alias is found in the Address Now database. **Y** Yes **N** No (default)
ReplaceAlias.Country	Specifies whether or not to overwrite the Country field if an alias is found in the Address Now database. **Y** Yes **N** No (default)
CautiousUpdate	This option, used in conjunction with the "Force Update", ensures that no major changes are made to the data during processing. **Y** Yes **N** No (default)
CrossComponentMatch	Specifies whether or not to correct common address standardization and validation errors by performing cross-component matching. Cross-component matching checks for matches between data found in one field in the input data and another field in the Address Now database. **Y** Yes **N** No (default)
UseReferenceDiacritics	Specifies whether or not ValidateGlobalAddress modifies the address to match the diacritics (accents, umlauts, etc.) in the postal database

optionName	Description/Valid Values
	when the only changes to the address are the diacritics . One of the following: **Y** Yes (default) **N** No For example, if **UseReferenceDiacritics** is enabled, the following would occur: Input City: Chalon-Sur-Saône City in the postal database: CHALON SUR SAONE Output City: CHALON SUR SAONE Input City: ARTEMIVS'K City in the postal database: ARTEMIVSK Output City: ARTEMIVSK If **UseReferenceDiacritics** is not enabled, the following would occur: Input City: Chalon-Sur-Saône Reference City: CHALON SUR SAONE Output City: Chalon-Sur-Saône Input City: ARTEMIVS'K Reference City: ARTEMIVSK Output City: ARTEMIVS'K Note that this option has no effect on the **Transliteration** option.
KeepStandardizationChanges	Specifies whether or not standardization changes such as changing "ROAD" to "RD" should be reported in the ACR code. **Y** Yes **N** No (default)
AcceptanceLevel	The Acceptance Level setting specifies the minimum number of address components that must be validated in order for the whole address to be considered validated. The value specified for AcceptanceLevel corresponds to the second character of the ACR code. For more information, see **The ACR Code** on page 289. The acceptance level differs from the InnerMatchScore option in that acceptance level measures how many components Validate Global Address validated, regardless of how well the validated components matched to address components in the postal databases, whereas InnerMatchScore indicates the probability that the output address is the correct, validated version of the input address. One of the following: **-1** The acceptance level is automatically set to an appropriate level based on the address's country. For example, U.S. addresses are processed with an acceptance level of 4.

optionName	Description/Valid Values
	0 No components validated (default) **1** Country only validated **2** City and country validated **3** City, postal code and country validated **4** Street, city, postal code and country validated **5** Premise number, building name, sub-building, PO box, company, street, city, postal code, and country validated
InnerMatchScore	Specifies the minimum confidence level for address validation. Addresses with a value in the Confidence output field greater than or equal to this value is validated, and those that have a lower value will not be validated (the output field Status will contain F.) Specify any value between 0 and 100. The higher the value, the higher the degree of confidence necessary for effective address validation. The default is 60.
CompanyWeight	A whole number from 0 to 10, indicating the relative importance of the FirmName field compared to the data in the Address Now database. This affects the confidence value, and can be used to tailor the confidence to distinguish correct and incorrect updates. For more information, see **The ACR Code** on page 289. The default value is 1.
StreetWeight	A whole number from 0 to 10, indicating the relative importance of the StreetName field compared to the data in the Address Now database. A whole number from 0 to 10, indicating the relative importance of this field compared to the others. For more information, see **The ACR Code** on page 289. The default value is 10.
CityWeight	A whole number from 0 to 10, indicating the relative importance of the City field compared to the data in the Address Now database. A whole number from 0 to 10, indicating the relative importance of this field compared to the others. For more information, see **The ACR Code** on page 289. The default value is 8.
PostcodeWeight	A whole number from 0 to 10, indicating the relative importance of the PostalCode field compared to the data in the Address Now database.

optionName	Description/Valid Values
	A whole number from 0 to 10, indicating the relative importance of this field compared to the others. For more information, see **The ACR Code** on page 289. The default value is 8
OuterMatchScoreLines	A value from 0 to 8 indicating the number of address lines to use when calculating the outer match score. The default is 8. For more information on the outer match score, see **The Outer Match Score** on page 289.

Output Format Options

Table 64: ValidateGlobalAddress Output Format Options

optionName	Description/Valid Values
FormatComponent.Department	Specifies whether or not the Department field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.FirmName	Specifies whether or not the FirmName field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.Building	Specifies whether or not the Building field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.SubBuilding	Specifies whether or not the SubBuilding field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.HouseNumber	Specifies whether or not the HouseNumber field should be included in the output of a formatted address.

optionName	Description/Valid Values	
	Y	Yes (default)
	N	No
FormatComponent.SubStreet	Specifies whether or not the SubStreet field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.StreetName	Specifies whether or not the StreetName field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.POBox	Specifies whether or not the POBox field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.SubCity	Specifies whether or not the SubCity field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.City	Specifies whether or not the City field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.USCountyName	Specifies whether or not the USCountyName field should be included in the output of a formatted address.	
	Y	Yes (default)
	N	No
FormatComponent.StateProvince	Specifies whether or not the StateProvince field should be included in the output of a formatted address.	

optionName	Description/Valid Values
	Y Yes (default) **N** No
FormatComponent.Principality	Specifies whether or not the Principality field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.PostalCode	Specifies whether or not the PostalCode field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.Plus4	Specifies whether or not the +4 field should be included in the output of a formatted address. **Y** Yes (default) **N** No
FormatComponent.Country	Specifies whether or not the Country field should be included in the output of a formatted address. **Y** Yes **N** No (default)
Transliteration	Specifies how to format diacritics in the output address. One of the following: **0** No transliteration is performed. Diacritic characters are left as specified in the input and/or postal database. Default. **1** Diacritic characters are removed and replaced with the equivalent unadorned character. **2** Diacritic characters are transliterated to an equivalent unadorned character or character sequence using language-specific transliteration rules. For example, the following shows the effect of each of the three transliteration options on a Swedish address. Note the differences in "Västra Frölunda". **0** Gustaf Wernersgata 12 S-42132 Västra Frölunda

optionName	Description/Valid Values
	1 Gustaf Wernersgata 12 S-42132 Vastra Frolunda **2** Gustaf Wernersgata 12 S-42132 Vaestra Froelunda

Output

Address Data Output

Table 65: ValidateGlobalAddress Address Data Output

columnName	Format	Description / Valid Values
AddressLine1	String [79]	The formatted first address line.
AddressLine2	String [79]	The formatted second address line.
AddressLine3	String [79]	The formatted third address line.
AddressLine4	String [79]	The formatted fourth address line.
AddressLine5	String [79]	The formatted fifth address line.
AddressLine6	String [79]	The formatted sixth address line.
AddressLine7	String [79]	The formatted seventh address line.
AddressLine8	String [79]	The formatted eighth address line.
ApartmentLabel	String [79]	Apartment designator (such as STE or APT). For example: 123 E Main St. APT 3
ApartmentNumber	String [79]	Apartment number. For example: 123 E Main St. APT **3**

columnName	Format	Description / Valid Values
Building	String [79]	The name of a building.
City	String [79]	The city name.
Country	String [79]	The ISO code or English name of the country. For a list of ISO codes, see **Country ISO Codes and Module Support** on page 908.
Department	String [79]	A subdivision of a country used in French and Spanish speaking countries. For example, France is divided into 100 departments.
FirmName	String [79]	The name of a company. For example: **Pitney Bowes Software** 4200 PARLIAMENT PL STE 600 LANHAM MD 20706-1844 USA
HouseNumber	String [79]	House number. For example: **123** E Main St. Apt 3
Latitude	String [79]	The most precise latitude that could be determined for the address. This could be a point level location or a centroid. The level of precision can be determined by looking at the ECRCode output field. For more information, see **The ECR Code** on page 288.
Longitude	String [79]	The most precise longitude that could be determined for the address. This could be a point level location or a centroid. The level of precision can be determined by looking at the ECRCode output field. For more information, see **The ECR Code** on page 288.
POBox	String [79]	The post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode	String [79]	The postal code. In the U.S. this is the ZIP Code™.
PostalCode.AddOn	String [79]	The 4-digit add-on part of the ZIP + 4® Code. For example, in the ZIP Code™ 60655-1844, 1844 is the 4-digit add-on. (U.S. addresses only.)

columnName	Format	Description / Valid Values
PostalCode.Base	String [79]	The 5-digit ZIP Code™. For example 20706 (U.S. addresses only.)
Principality	String [79]	An area within a country. For example, England, Scotland, and Wales are principalities. This field will normally be blank.
StateProvince	String [79]	The state or province abbreviation.
StreetName	String [79]	Street name. For example: 123 E **Main** St. Apt 3
StreetSuffix	String [79]	Street suffix. For example: 123 E Main **St**. Apt 3
SubCity	String [79]	A district or suburb. The subcity is used in countries where it is common to include the district or suburb within the address. For example, 27 Crystal Way **Bradley Stoke** Bristol BS32 8GA In this case, "Bradley Stoke" is the subcity.
SubStreet	String [79]	The second street address used to identify an address. Substreets are used in countries where it is common to give two street names in the address. For example, 12 The Mews **High Street** In this example, "High Street" is the substreet. Substreets can be used to precisely identify the delivery location. In the example, "The Mews" may be a small street that needs another street identification to properly locate the address, so "High Street" is included. In this case, "High Street" is the main or known street.
USCountyName	String [79]	For U.S. addresses, the name of the county where the address is located.

Return Codes

Table 66: ValidateGlobalAddress Return Codes

columnName	Format	Description / Valid Values
ACRCode	String [79]	The Address Correction Result (ACR) code describes what data has been changed in each record. For information on what this code means, see **The ACR Code** on page 289.
Confidence	String [79]	The level of confidence assigned to the address being returned. Range is from zero (0) to 100; zero indicates failure, 100 indicates a very high level of confidence that the match results are correct. This value is the same as the last three digits of the ACR code, referred to as the validation match score. For more information, see **The ACR Code** on page 289.
ECRCode	String [79]	The Enhanced Correction Result (ECR) code describes the level of precision of the latitude and longitude returned for the address. For mroe information, see **The ECR Code** on page 288.
Email1	String [79]	Extra standardization information.
Email2	String [79]	Extra standardization information.
OuterMatchScore	String [79]	A score that measures changes to each address line. For more information, see **The Outer Match Score** on page 289.
Status	String [79]	Reports the success or failure of the match attempt. • **null**—Success • **F**—Failure
Status.Code	String [79]	Reason for failure, if there is one. • UnableToValidate • ServerError • CountryNotFound
Status.Description	String [79]	Description of the problem, if there is one. • **Address Not Found**—This value will appear if Status.Code=UnableToValidate.

columnName	Format	Description / Valid Values
		• **Module not licensed**—This value will appear if Status.Code=ServerError. • **Could Not Identify Country**—This value will appear if Status.Code=CountryNotFound.
URL1	String [79]	Extra standardization information.
URL2	String [79]	Extra standardization information.
WCRCode	String [79]	The Word Correction Result (WCR) code describes vulgar words found in the input address. The code has two components: • **Location code**—One of the following: • **AB**—Indicates a vulgarity was found in the address. • **NB**—Indicates a vulgarity was found in the name. • **Count**—The number of vulgar words found in the location indicated by the location code. For example, AB2 indicates that two vulgar words were found in the input address.

The ECR Code

The Enhanced Correction Result (ECR) code describes the level of precision of the latitude/longitude coordinates returned for the address. The code consists of a prefix followed by a dash then the body of the code.

The prefix always begins with "EL" followed by a number from 1 to 5 indicating the overall level of precision:

- 5—Point geocode
- 4—Street centroid
- 3—Postcode centroid
- 2—City centroid
- 1—Region centroid

The body of the code identifies the components that were used to match the address to a geocode. Note that the body consists of letters and numerals. Here's what they mean:

- P—Premise/house number, building or PO box
- S—Street
- T—City
- R—Region/state
- Z—Postal code

- C—Country

There are only two numeric options in the body: 4 or 0

- 4—The component data was available to make the geocode-address match.
- 0—The component date was not available.

For example: EL4-P0S4T4R4Z4C4

In this example, the 0 following the P tells us that premise/house number data was not available to make this address match; everything from street to country, however, was used in the assignment.

The Outer Match Score

The outer match score indicates how much ValidateGlobalAddress changed each address line to validate the address. The score compares the address lines before standardization and after validation and formatting. This score is only generated if you set the option `OuterMatchScoreLines` to a value greater than 0.

The outer match score is similar to the validation match score, which is part of the ACR code (see **The ACR Code** on page 289). The difference is that the outer match score measures any change to an address line, including formatting, whereas the validation match score measures only whether or not the data could be validated,

For example, take the following input address lines before processing:

Address Line 1: 5 camden cres Address Line 2: bath Address Line 3: uk

After processing the address lines are:

Address Line 1: 5 Camden Crescent Address Line 2: Bath Address Line 3: BA1 5HY Address Line 4: United Kingdom

This has a validation match score of 84% and in outer match score of 23%.

The validation match score is high because the address components were fairly accurate before validation. The street name was valid except for casing and use of an abbreviation. The city and country were both valid. The only thing not correct was the postal code (in this case it was missing). Hence the relatively high validation match score of 84%.

The outer match score is low because after formatting, the address lines are considerably different from the input. In this case, Address Line 3 contained "uk" on input, and contains "BA1 5HY" on output. Line 4 was empty on input, and in populated on output. Address line 1 has also changed. The outer score is therefore quite low.

The ACR Code

The Address Correction Result (ACR) code describes what data has been changed in each record. An example of an ACR is:

L5-P0S0A5T1R0Z0C4-098

ACR codes consist of three parts:

- Validation Level

- Component Status
- Validation Match Score

Validation Level

The first two characters of the address correction result state the type and level of validation.

The first character, which is always alphabetic, specifies the type of validation:

- **U**—Unable to standardize address
- **C**—Address is in component form
- **L**—Address has been formatted into address lines
- **R**—Address has been reverted and has not reached acceptable level

The second character, which is always numeric, specifies the level of validation. The higher the level, the better the validation will be. The levels that can be achieved are as follows:

- **0**—No components validated
- **1**—Country only validated
- **2**—City and country validated
- **3**—City, postal code and country validated
- **4**—Street, city, postal code and country validated
- **5**—Premise number, building name, sub-building, PO box, company, street, city, postal code, and country validated.

Component Status

The second part of the ACR code gives the status of the main address components. The address components are identified as follows:

- Character 3-4: P—Premise/house number
- Character 5-6: S—Street
- Character 7-8: A—Subcity (city area)
- Character 9-10: T—City
- Character 11-12: R—Region/state
- Character 13-14: Z—Postal code/ZIP Code®
- Character 15-16: C—Country

A number follows each component and can take one of the following values:

- **0**—Not found/empty
- **1**—Derived using position in input data
- **2**—Recognized using the Address Now Module database
- **3**—Recognized and updated to standard form using the Address Now Module database
- **4**—Validated using Address Now Module database
- **5**—Updated/corrected using Address Now Module database
- **6**—Added using Address Now Module database
- **7**—Correctly empty
- **8**—Partial recognition using Address Now Module database

- **9**—Needs correcting to match Address Now Module database

Validation Match Score

The Validation Match Score comprises characters 17-19, the final three digits of the ACR code. This is a comparison between the standardized data (in component format) and the suggested match returned from the Address Now Module database.

This score is calculated by examining all fields returned from the Address Now Module database and comparing them individually with the existing component data. The overall match score is then calculated by combining these individual values into an average score, taking into account the match score weightings, which can be set from the address validation options dialog box. For example,

Input data:

AddressLine1: 11 High Street
City: Anytown
Country: UK

Standardized data:

Premise: 11
Street: High Street
City: Anytown

When validated, the data returned from the Address Now Module database for this record may be:

Premise: 11
Street: High Street
City: Anytown
Postal Code: ZZ9 9ZZ

Comparing the Address Now Module database to the standardized data we get:

- Premise: 100% match
- Street: 100% match
- City: 100% match
- Postal Code: not used, because empty on input

Combining these percentages gives us a match score of 100%.

Another example may be:

Input data:

AddressLine1: bergerstrasse 12
AddressLine2: munich
AddressLine3: 80124
Country: Germany

Standardized data:

Premise: 12
Street: Bergerstr.

City: München
Postal Code: 80124

Address Now Module database output:

Premise: 12
Street: Burgerstr.
City: München
Postal Code: 80142

Comparing the Address Now Module database output to the standardized data we get:

- Premise:100% match
- Street: 90% match (the actual figure is determined by a textual comparison of the two values)
- City: 100% match
- Postal Code: 80% match (because the numbers are transposed)

This gives an overall match score of 92% if the match score weightings are all set at 1. Increasing the match score weighting of the postal code will decrease the match score, because the postal code component score (80%) will be made more important in the calculation. Increasing the match score weighting of the city will increase the match score, because the city component score (100%) will be made more important.

For example:

L5-P4S4A5T5R4Z4C4-098

- L shows that formatting has been carried out to create the address lines
- The validation level is 5, meaning that the highest level of matching against the Address Now Module database was attained
- All component codes except subcity (A) and city (T) are set to 4 indicating that they were validated using the Address Now Module database
- The subcity code and city code are set to 5 indicating that these components were corrected using the Address Now Module database

The overall address matched the Address Now Module database at 98%.

Note: You may also receive a value of "SDS" for the Validation Match Score. A return of SDS indicates that the address has not been standardized, possibly as a result of the address being reverted.

12

Enterprise Geocoding Module

In this section:

- What is the Enterprise Geocoding Module? 294
- Country Identifier 304
- GeocodeAddressAUS 305
- GeocodeAddressGBR 332
- GeocodeAddressGlobal 350
- GeocodeAddressWorld 426
- GeocodeUSAddress 464
- GNAFPIDLocationSearch 515
- ReverseAPNLookup 526
- ReverseGeocodeAddressGlobal 539
- ReverseGeocodeUSLocation 551
- Geocode US Address Auxiliary Files 568
- Location Codes for U.S. Geocoding 574
- Match Codes for U.S. Geocoding 592
- Result Codes for International Geocoding 597
- Custom Databases for International Geocoding 602
- Encountering False Positives 609

What is the Enterprise Geocoding Module?

The Enterprise Geocoding Module performs address standardization, address geocoding, and postal code centroid geocoding. You can enter an address and get outputs such as geographic coordinates, which can be used for detailed spatial analysis and demographics assignment. You can also enter a geocode, a point represented by a latitude and longitude coordinate, and receive address information about the provided geocode.

Enterprise Geocoding Components

Enterprise Geocoding Module consists of the following components. The specific components you have depend on your license.

- **GeocodeAddressAUS**—Takes an address in Australia and returns latitude/longitude coordinates and other information.
- **GeocodeAddressGBR**—Takes an address in Great Britain and returns latitude/longitude coordinates and other information.
- **GeocodeAddressGlobal**—Takes an address in any supported country and returns latitude/longitude coordinates and other information. This component will only geocode addresses from countries you have licensed. It does not support Australia and Great Britain.
- **GeocodeUSAddress**—Takes an input address and returns latitude/longitude coordinates and other address information.
- **GNAFPIDLocationSearch**—Identifies the address and latitude/longitude coordinates for a Geocoded National Address File Persistent Identifier (G-NAF PID).
- **ReverseAPNLookup**—Takes an Assessor's Parcel Number (APN), Federal Information Processing Standards (FIPS) county code, and FIPS state code and returns the address of the parcel.
- **ReverseGeocodeUSLocation**—Takes as input a geocode (latitude and longitude coordinate) and returns the address of the location.

Enterprise Geocoding Databases

The following table lists the Enterprise Geocoding Module databases. The databases are installed on the Spectrum™ Technology Platform server. Some of the databases are available by subscription from Pitney Bowes Software and are updated monthly or quarterly. Others are licensed from the USPS®.

Table 67: Enterprise Geocoding Module Databases

Database Name & Description	Required or Optional	Supplier
U.S. Geocoding Databases These databases contain the spatial data necessary to perform address standardization and geocoding. You must	Required for U.S. geocoding	Pitney Bowes Software monthly subscription

Database Name & Description	Required or Optional	Supplier
install at least one of these databases. You set the database that you want to match against with the processing options. Enterprise Geocoding tries to match to the database you indicate. To verify you are matching to the database you want, you can review the value returned in the StreetDataType output field. These databases use proprietary files called GSD files. For ZIP Code centroid matching, the file us.Z9 contains all the centroid info for all states and normally has a z9 extension. • **Centrus Enhanced Geocoding**—This database consists of TIGER data provided by the U.S. Geological Survey and address data provided by the U.S. Postal Service. • **GDT Geocoding**—This database provides more up-to-date data than the Centrus Enhanced Geocoding database. It requires an additional license. GDT data is provided by TomTom, a third-party provider of spatial data, and postal data from the U.S. Postal Service. • **NAVTEQ Geocoding**—This database provides more up-to-date data than the Centrus Enhanced Geocoding database. It requires an additional license. NAVTEQ data is provided by NAVTEQ, a third-party provider of spatial data. For more information about these databases, contact your sales representative. • **ZIP + 4 Centroid**—This database provides only address standardization and ZIP + 4 centroid matching. It does not provide street-level matching. Each geocoding database has an optional Statewide Intersections Index. The Statewide Intersection Index is designed to enable fast intersection identification on a statewide basis. For example, the Statewide Intersection Index will allow the database search for "1st and Main St, CO" and return a list of possible matches in Colorado more quickly than searching the entire geocoding database for each instance of the intersection.		
International Geocoding Databases International geocoding databases contain the spatial data necessary to perform address standardization and geocoding for locations outside the U.S. Each country has its own database, and some countries have optional databases that provide enhanced geocoding.	Required for all licensed countries	Pitney Bowes Software quarterly subscription

Database Name & Description	Required or Optional	Supplier
Australia Geocoded National Address File (G-NAF) This database provides enhanced geocoding for Australian addresses. This is the only authoritative Australian national index of locality, street and number, validated with geographic coordinates. It contains both officially recognized rural and urban addresses and unofficial addresses (aliases). Postal addresses and PO Boxes are not included. However, because some rural areas do not have adequate rural address information, roadside mail box (RMB) numbers, Lot numbers, and Block & Section numbers have been included in the G-NAF data set. When you install this database there will be two subfolders: • **GNAF123**—Contains the point-level dictionary. This has the highest precision of geocoding (characterized by Reliability Level 1, 2, or 3.) • **GNAF456** —Contains the remainder of address information in G-NAF that does not meet high precision geocoding criteria (characterized by Reliability Level 4, 5, or 6.) You must specify each of these as separate database resources in the Management Console. We recommend that you use both databases to validate the existence of addresses but only use the GANF123 for parcel-level geocoding. If you do not require parcel-level geocodes you can use the GANF456 database for geocoding.	Optional	Pitney Bowes Software quarterly subscription
United Kingdom Address-Point Database The Address-Point database contains street addresses and is derived from the Ordnance Survey®. Candidate location will be usually to a resolution of 0.1 meters from the building/parcel delivery point. This database is suitable for applications that demand this highest available level of precision, including routing applications. The Address-Point database provides a greater level of geocoding precision than can be delivered by the CodePoint database. This high level of precision is reflected in S8 and S7 result codes. For more information on the Ordnance Survey Address-Point data source, see: **www.ordnancesurvey.co.uk/oswebsite/products/addresspoint/**	Optional	Pitney Bowes Software

Database Name & Description	Required or Optional	Supplier
While the Ordnance Survey data source does not contain addresses for Northern Ireland, the Address-Point dataset from Pitney Bowes Software is supplemented with Royal Mail® postcode address data for Northern Ireland. This Northern Ireland data has postcode centroid (result code S3) precision only.		
United Kingdom CodePoint Database The CodePoint Postal Address File (PAF) database provides postcode centroid geocoding. The CodePoint database is suitable for most applications involving address matching, validation, etc. The CodePoint database is derived from Royal Mail and covers street addresses for the UK (Great Britain and Northern Ireland). The CodePoint database is licensed for the entire dataset, rather than by region. For more information on the Royal Mail data source, see: **www.royalmail.com** The postcode centroid precision provided by the CodePoint database is reflected in S3 result codes.	Optional	Pitney Bowes Software
Reverse Geocoding Database (U.S. Only) This database contains the data you need to convert a latitude/longitude location to an address.	Optional, but required for ReverseGeocodeUSLocation. Additional license required.	Pitney Bowes Software monthly subscription
New Zealand Point Database The New Zealand Point Database is based on postal point data which has a roof top precision point of each unique street address. Location X and Y returned for candidates from this database are roof top precision. This data is maintained by the government authority, Land Information New Zealand. This database is a monthly update from what the local district councils supply.	Optional	Pitney Bowes Software
U.S. Points Databases Points databases contain data for locating the center of a parcel. These databases provides enhanced geocoding	Optional, but ReverseAPNLookup requires Centrus Enhanced Points or	Pitney Bowes Software monthly subscription

Database Name & Description	Required or Optional	Supplier
accuracy for internet mapping, property and casualty insurance, telecommunications, utilities, and others. • **Centrus Points**—This database contains the data necessary to locate the center of a parcel or building. It does not contain assessor's parcel number (APN) or elevation data. • **Centrus Elevation**—This database contains the same data as Centrus Points, plus elevation data. • **Centrus Enhanced Points**—This database contains the same data as Centrus Points, plus APN data. • **Centrus Premium Points**—This database contains the same data as Centrus Points, plus both APN and elevation data. • **Centrus TomTom Points Database**—The data in this database is provided by TomTom, a third-party provider of spatial data.	Centrus Premium Points. Additional license required.	
Auxiliary Files Auxiliary files contain user-defined records. You can use auxiliary files to provide custom data to use in address matching and geocode matching. For more information, see **Auxiliary File Overview** on page 568.	Optional	User-defined
DPV® Database (U.S. Only) The Delivery Point Validation database allows you to check the validity of any individual mailing address in the U.S. (over 145 million locations). The DPV database is distributed as an optional feature and can be installed to enhance the geocoding database's ability to validate mailing addresses. Each time an edition of the geocoding database is released, a corresponding edition of the optional DPV database is released. The date of the DPV database must match the date of the geocoding database for DPV processing to function. DPV lookups may not be performed after the expiration date of the DPV database. For more information on DPV, see **Delivery Point Validation (DPV)** on page 303. **Note:** CASS processing requires the DPV option. The DPV option is also required to determine ZIP + 4 and ZIP + 4 related output (DPBC, USPS record type, etc.).	Optional, but required for CASS Certified™ processing. Additional license required.	Pitney Bowes Software monthly subscription

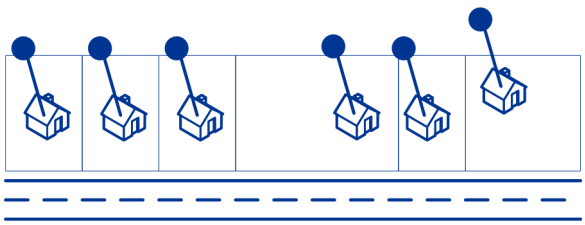
Database Name & Description	Required or Optional	Supplier
Postal Service licensing prohibits using DPV for the generation of addresses or address lists, and also prohibits the DPV database being exported outside the United States.		
EWS Database (U.S. Only) The Early Warning System (EWS) database contains data that prevents address records from miscoding due to a delay in postal data reaching the U.S. Postal database. For more information, see **Early Warning System (EWS)** on page 303. The USPS® refreshes the EWS file on a weekly basis. Unlike the DPV and LACS^{Link} databases, the EWS database does not need to have the same date as the geocoding database. You can download the EWS file from the CASS section of the USPS® RIBBS website at: **www.ribbs.usps.gov** When you download the EWS database, you will receive a file named OUT. You must rename the OUT file to EWS.txt before using it.	Optional	Download for free from USPS® website
LACS^{Link} Database (U.S. Only) The LACS^{Link} database allows you to correct addresses that have changed as a result of a rural route address converting to street-style address, a PO Box renumbering, or a street-style address changing. For more information, see **Locatable Address Conversion System (LACS)** on page 303. The date of the LACS^{Link} database must match the date of the geocoding database for LACS^{Link} processing to function. **Note:** The Enterprise Geocoding Module requires the LACS^{Link} option in CASS mode to receive ZIP + 4 and ZIP + 4 related output (delivery point barcode, USPS record type, etc.). USPS licensing prohibits using LACS^{Link} for the generation of addresses or address lists, and also prohibits the LACS^{Link} database being exported outside the United States.	Optional, but required for CASS Certified™ processing	Pitney Bowes Software monthly subscription

Geocoding Concepts

Geocoding is the process of determining the latitude/longitude coordinates of an address. There are different ways that an address can be geocoded. In order of most accurate to least accurate, these methods are:

Point Level Matching

Point-level matching locates the center of the actual building footprint or parcel. This is the most accurate type of geocode and is used in industries such as internet mapping, insurance, telecommunications, and utilities.

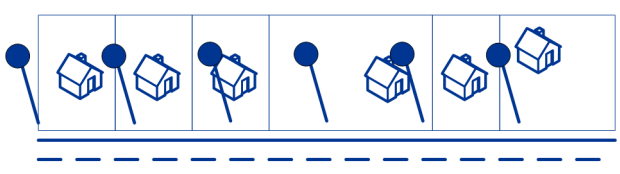


Centerline matching is used with point-level matching to tie a point-level geocode with its parent street segment. This provides you with additional data about the parent street segment that is not retrievable using only the point-level match. The output information also includes the bearing from the point data geocode to the centerline match.

Street Matching

Street matching identifies the approximate location of an address on a street segment. In street matching, the location is determined by calculating the approximate location of a house number based on the range of numbers in the location's street. For example, if the address is on a street segment with a range of addresses from 50 to 99, then it is assumed that the house number 75 would be in the middle of the street segment. This method assumes that the addresses are evenly spaced along the street segment. As a result, it is not as exact as point matching since addresses may not be evenly distributed along a street segment.

For example, the following diagram shows the results of street-level matching along a segment with unevenly-spaced buildings. The first three buildings are fairly accurately geocoded because they are evenly spaced. The fourth building, however, resides on a slightly larger parcel than the others along this street. Since street-level matching assumes that the buildings are evenly spaced, the result is that fourth, fifth, and sixth houses are not as precise as the first three. If you were to use point-level geocoding, the results would be more accurate.

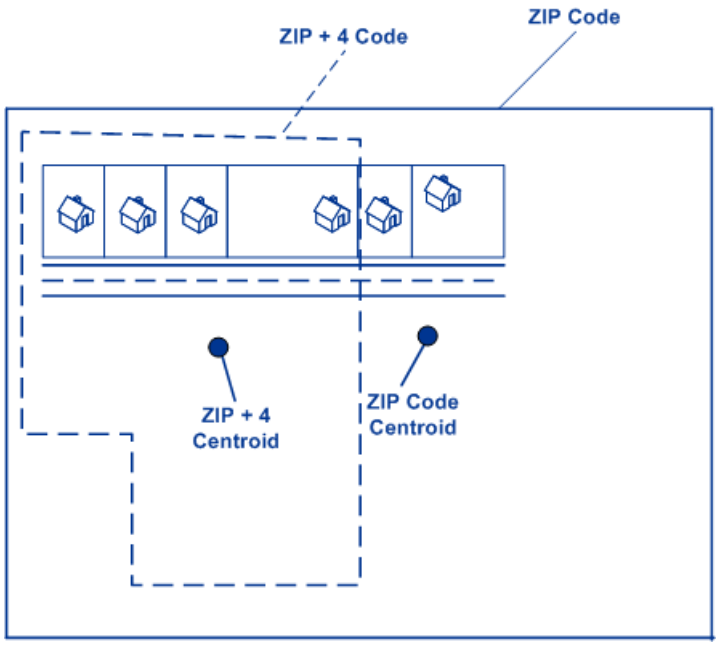


Centroid Matching

ZIP Code centroid matching is a center point of an area defined by either a ZIP Code or a ZIP + 4, and is the least accurate type of geocode. A ZIP Centroid is the center of a ZIP Code; a ZIP + 4 centroid is

the center of a ZIP + 4. Since a ZIP + 4 represents a smaller area than a ZIP Code, a ZIP + 4 centroid is more accurate than a ZIP Code centroid.

The following diagram illustrates centroid matching. All six houses would have the same geocode in this example because they all reside in the same ZIP + 4 code.



Geocoding Match Strategies for Non-U.S. Locations

The Enterprise Geocoding Module offers a variety of options for controlling geocoding precision and match rate. The following information describes different approaches for matching which you can apply to any country geocoder except the U.S. geocoder (GeocodeUSAddress), which has a different set of options.

Maximizing the Match Rate

To generate the highest match rate possible, do not specify house number, street, and city/locality using the ExactMatch option.

Another way to maximize the match rate is by setting FallbackToPostal=Y. This means that the geocoder will fall back to the four-digit postcode centroid if a close street level match cannot be made. While this scenario might yield false positives, it may be the best matching solution when you have large databases to geocode.

You should evaluate if the percentage of false positives will affect your analysis. To reduce the number of false positives without sacrificing hit rate, analyze the result codes after a geocoding session and adjust your settings accordingly.

Maximizing Precision

If your analysis requires highly precise geocoded addresses, choose a strategy in which the geocoder returns the maximum percentage of high precision geocodes and the lowest number of imprecise matches (false positives). To do this, use the ExactMatch option to require close matches to match on all address elements. Also, set FallbackToPostal=N.

This technique may produce a lower percentage match rate, but will provide the best precision.

Balancing Match Rate and Precision

You may want to use a balanced strategy between match rate and geographic precision. That is, you may want to geocode as many records as possible automatically, but at the same time want to minimize the number of weaker matches (false positives). Some examples of false positives can occur when the geocoder:

- Finds a street that sounds like the input street
- Finds the same street in another city (if postal code is relaxed)
- Finds the street but with a different house number (if house number is relaxed)

To achieve a good balance between match rate and precision, use the following settings:

- **CloseMatchesOnly**—Specify "Y".
- **MustMatchHouseNumber**—Specify "Y".
- **MustMatchStreet**—Specify "Y".
- **Fall backToPostal**—Specify "N".

Postal Concepts

The following sections contain information on postal concepts used by the Enterprise Geocoding Module.

Dual Addresses

GeocodeUSAddress can process input that contains two addresses for the same record on the same address line. For example, GeocodeUSAddress can process the following input address:

3138 HWY 371
PO BOX 120
PRESCOTT AR 71857

GeocodeUSAddress does not recognize dual addresses where the two addresses are both street addresses. For example, GeocodeUSAddress does NOT recognize 135 Main St 4750 Walnut St Ste 200. GeocodeUSAddress does recognize dual addresses where the two addresses are the same type of address but are not street addresses. For example, GeocodeUSAddress does recognize PO BOX 12 PO BOX 2000.

After GeocodeUSAddress parses the dual address, it searches for a match. GeocodeUSAddress determines which address has preference for a match based on the processing mode. In CASS mode, GeocodeUSAddress ignores the prefer PO Box and prefer street options, and attempts to find a match based on the following order: PO Box, Street, Rural Route, and General Delivery. In Relaxed mode, GeocodeUSAddress recognizes the Address Preference (AddressPreference) input option.

Note: GeocodeUSAddress does not perform dual address processing in Exact and Close mode. GeocodeUSAddress does not perform dual address processing on multi-line addresses.

Locatable Address Conversion System (LACS)

The USPS® Locatable Address Conversion System (LACS) corrects addresses that have changed as a result of a rural route address converting to street-style address, a PO Box renumbering, or a street-style address changing. The following are examples of LACS^{Link} conversions:

- Rural Route Converted to Street-Style Address: Old Address: RR 3 Box 45 New Address: 1292 North Ridgeland Drive
- Street Renamed and Renumbered: Old Address: 23 Main Street New Address: 45 West First Avenue
- PO Box Renumbered: Old Address: PO Box 453 New Address: PO Box 10435

LACS^{Link} is required for CASS processing.

Delivery Point Validation (DPV)

Delivery Point Validation (DPV®) is a United States Postal Service® (USPS®) technology that validates the accuracy of address information down to the individual mailing address. By using DPV® to validate addresses, you can reduce undeliverable-as-addressed (UAA) mail, thereby reducing postage costs and other business costs associated with inaccurate address information.

Without DPV®, the address validation process only verifies that an individual address is within a range of valid addresses for the given street. For example, the USPS data indicates that the range of addresses on Maple Lane is 500 to 1000. You attempt to validate an address of 610 Maple Ln Without DPV®, this address would appear to be valid because it is in the range of 500 to 1000. However, in reality the address 610 Maple Ln does not exist: the house numbers in this section of the street are 608, 609, 613, and 616. With DPV® processing, you would be alerted to the fact that 610 Maple Ln does not exist and you could take action to correct the address.

DPV® also provides unique address attributes to help produce more targeted mailing lists. For example, DPV® can indicate if a location is vacant and can identify commercial mail receiving agencies (CMRAs) and private mail boxes.

Although DPV® can validate the accuracy of an existing address, you cannot use DPV® to create address lists. For example, you can validate that 123 Elm Street Apartment 6 exists, but you cannot ask if there is an Apartment 7 at the same street address. To prevent the generation of address lists, the DPV® database contains false positive records. False positive records are artificially manufactured addresses that reside in a false positive table. For each negative response that occurs in a DPV® query, a query is made to the false positive table. A match to this table will stop DPV® processing.

Early Warning System (EWS)

The Early Warning System (EWS) provides up-to-date address information for new and recently changed addresses that have not yet been updated in the monthly USPS database. EWS prevents address records from miscoding due to a delay in postal data reaching the USPS® databases.

The older the U.S. Postal Database, the higher potential you have for miscoding addresses. When a valid address is miscoded because the address it matches to in the U.S. Postal Database is inexact, it will result in a broken address.

EWS data consists of partial address information limited to the ZIP Code™, street name, predirectional, postdirectional, and a suffix. For an address record to be EWS-eligible, it must be an address not present on the most recent monthly production U.S. Postal Database.

The USPS® refreshes the EWS file on a weekly basis. You can download the EWS file from the USPS® website at **ribbs.usps.gov/files/CASS**.

Country Identifier

What is Country Identifier?

Country Identifier stage takes a country name or a combination of postal code and state/province and returns the two-character ISO country code, the three-character Universal Postal Union (UPU) code, and the English version of the country name. Use this stage to determine the country codes for an address before geocoding or to validate an existing country code.

Input

There are two ways to specify input to Country Identifier:

- A value in the Country field
- Values in both the StateProvince and PostalCode fields

If you include Country, StateProvince, and PostalCode fields, Country Identifier first attempts to determine the country using the value in the Country field. If it cannot determine the country, it then uses the StateProvince and PostalCode fields.

Table 68: Country Identifier Input Data on page 304 lists the input fields for Country Identifier.

Table 68: Country Identifier Input Data

columnName	Format	Description / Valid Values
City	String	The name of the municipality where the address is located.
Country	String	The country code or name, in any of the following formats: • Two-character ISO country code • Three-character UPU country code • The country name (for example, Spain, Espana, or España) For a list of the ISO codes, UPU codes, and English names, see **Country ISO Codes and Module Support** on page 908.
PostalCode	String	The postal code for the address.
StateProvince	String	The state or province where the address is located.

Options

Country Identifier stage has no options.

Output

The following table lists the output fields for Country Identifier.

Table 69: Country Identifier Output Data

columnName	Description
City	The value specified in the input field City.
Country	The value specified in the input field Country.
Country.ISO	The two-character International Organization for Standardization (ISO) code for the country. For a list of the ISO codes, see **Country ISO Codes and Module Support** on page 908.
Country.Name	The English version of the country name. For a list of English names, see **Country ISO Codes and Module Support** on page 908.
Country.UPU	The three-character Universal Postal Union (UPU) code for the country. For a list of the UPU codes, see **Country ISO Codes and Module Support** on page 908.
PostalCode	The value specified in the input field PostalCode.
StateProvince	The value specified in the input field StateProvince.

GeocodeAddressAUS

GeocodeAddressAUS takes an address or street intersection in Australia and returns the geographic coordinates. GeocodeAddressAUS can geocode to street addresses and postal code centroids.

GeocodeAddressAUS is an optional part of the Enterprise Geocoding Module. For more information about Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Input

Input Fields

GeocodeAddressAUS takes an address or intersection as input. To obtain the best performance and the most possible matches, your input address lists should be as complete as possible, free of misspellings and incomplete addresses, and as close to postal authority standards as possible. Most postal authorities have websites that contain information about address standards for their particular country.

The following table lists the input fields used for geocoding locations in Australia.

Note: Specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 70: Input Fields for Australia

columnName	Description / Valid Values
AddressLine1	One of the following: • The address line containing the street name and building number. For example: **4360 DUKES RD** KALGOORLIE WA 6430 • The full address. For more information, see **Single Line Input** on page 309 • A street intersection. To specify a street intersection, use double ampersand (&&) to separate the streets. For more information, see **Street Intersection Input** on page 309.
City	The city or town name. Your input address should use the official city name.
County	The Local Government Authority (LGA) name.
FirmName	Company name. For example: **Pitney Bowes** 4360 Dukes Rd Kalgoorlie WA 6430
HouseNumber	The building number. You may get better parsing results for some countries if you put the house number in this field instead of AddressLine1. For the best results, use the HouseNumber input field

columnName	Description / Valid Values
	for these countries: India, Singapore, Malaysia, Thailand, and Turkey when using multi-line input. **Note:** The house number specified in the HouseNumber field takes precedence over any house number specified in the AddressLine1 field.
LastLine	The last line of the address. 4360 DUKES RD **KALGOORLIE WA 6430**
PostalCode	The postal code in the appropriate format for the country. Australia uses a four-digit postal code system. In general, the first digit represents a state or territory, the second digit represents a region with a state, and digits three and four representing towns. For more information, see **Address Guidelines for Australia** on page 307.
StateProvince	The state name.

Address Guidelines for Australia

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Australia addresses, see the Australia Post website: **www.auspost.com.au**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **House numbers and apartments**—Remove spaces between house number and apartment letter. 123 A Main Street does not geocode correctly because the geocoder assumes that the name of the street is A. Two workaround options are available:
  - Do not include the apartment letter.
  - Delete the space between house number and apartment letter: 123A Main Street geocodes because the geocoder ignores the A.
- **House numbers and unit information**—The house number pinpoints the location of the address. Unit input can be in one of two formats, as shown in the following examples:
  - Flat 2, 17 Jones St.
  - Apt 19, 123 Main St., where Apt is the unit type and 99 is the unit number. In this format, you must specify a valid unit type, otherwise the address will not be geocoded correctly. For a listing of valid unit types, see **www.auspost.com.au**.
  - 99-123 Main St. For an address derived from the G-NAF database, this address is a unique house number and is geocoded as a single delivery point, not as a range.

- **Directional suffixes**—Use directional suffixes wherever possible. This is especially true in towns and cities that consist mainly of numbered streets. These streets can only be distinguished by their directional suffixes and street types. They also distinguish addresses on streets that change direction. For example: 123 Queen St W and 123 Queen St E would have very different coordinates.
- **Street types**—These distinguish different streets of the same name. For example, Main Avenue and Main Street are two entirely different entities. Using types is not essential, but it adds precision to your data. For a listing of street types, see **www.auspost.com.au**.
- **City/suburb name**—Enter the city/suburb name in the City field. Note that all input addresses must contain either a city or a postal code. If the geocoder does not make a close match on the street in the specified suburb, it can make a close match on the Local Government Area (LGA). Local Government Area (LGAs) do not encompass all of Australia. For example, LGAs do not cover extensive northern parts of South Australia, a large part of the Northern Territory, and the Australian Capital Territory. An LGA can include a number of official suburbs. It is best to use the suburb name for geocoding purposes, but it is possible to get a match on the LGA (or to return LGA information) in geocoded results.
- **Postal code**—All postcodes consist of four digits. While there are exceptions, the general format of postcodes is as follows:

  Note: These are general guidelines, and there are exceptions to these postcode number ranges.

  - Digit 1 represents the state or territory, within the following general conventions:

2	NSW
2600 and 2900	ACT
3	VIC
4	QLD
5	SA
6	WA
7	TAS
0	NT

  - Digit 2 represents a region within the state. State or territorial capitals usually have a 0 or 1 as the second digit.
  - Digits 3 and 4 represent towns. Major towns usually have a 0 as the last digit (or sometimes as the last two digits).

Single Line Input

Instead of entering each address element in separate fields, you may enter the entire address in the AddressLine1 input field.

You can enter addresses in this single-line format:

```
StreetAddress;City;StateProvince;PostalCode
```

Note: Australia addresses must be in this format. You must include either a city plus state/province or a postal code as the last address element.

Where:

- *StreetAddress* can be house number and street name in either order (with street type immediately before or after the street name).
- *City* is the town.
- *StateProvince* is the postal abbreviation for the state or province.
- *PostalCode* is the complete postcode.

The matching accuracy for single line input is comparable to that of structured address input. The performance of single line input addresses may be slightly slower than that of structured address input.

For best results, use delimiters (comma, semicolon, or colon) between each component of the address. For example,

```
18 Merivale St,South Brisbane,QLD,4101
```

If the input address is missing delimiters, spaces are recognized as separators and internal parsing rules identify address components. In the example above, the address would still successfully geocode even if some or all of the delimiters were missing in the input.

Note: Non-delimited or partially-delimited single line addresses may take longer to geocode and may not produce the same results as delimited single line input. This is especially true for addresses with multi-word street names or cities. To optimize single line geocoding, use delimiters between address components (particularly between street name and city).

Punctuation is ignored for geocoding purposes.

Guidelines for Single Line Input

- The country is not required. Each country geocoder assumes that the address is in its country.
- Firm information (placename, building name, or government building) is returned if available.

Street Intersection Input

If you enter a street intersection as input, the geocoder will provide the coordinates of the intersection.

To enter an intersection, specify the two street names separated by a double ampersand (&&) in AddressLine1. For example:

AddressLine1: `Ocean Ave && New South Head Rd`
City: `Woollahra`

Note: Do not use /, _, -, or any other separator. Intersections must be delimited using a double ampersand (&&) as a separator.

All close match criteria are enforced for intersection geocoding, just as for any street level geocoding.

Address Range Input

Street range data represents a range of house numbers that can possibly exist but are not guaranteed to exist. For addresses derived from the Street Range Address database, house number matching is more relaxed compared to the G-NAF database. Also, the candidate house number may be changed based on how the input house number matches the suffix/range data.

The following table shows geocoding match results with house number ranges from a Street range data source:

Input House Number	Candidate House Number Range	Candidate House Number	Match or Non-Match
10	10-12	10	Match
10A	10-12	10	Match
10	8-12	10	Match
10-14	10-12	10	Match
10-14	10-20	10-14	Match
10	12-16	12	Non-Match

G-NAF Range Address Matching

Australian addresses originating from the G-NAF database may contain house number ranges, but these records still represent single address delivery points. These range addresses may also have alphabetic suffixes. For example, the following house address numbers each represent a single address.

10-12 10A-10C 10-10A

GeocodeAddressAUS validates and geocodes these point source addresses. If the criteria are met and the reliability of the match is 1 or 2, GeocodeAddressAUS returns point matches with an S8 result code. See the description of the output field AUS.GNAF_Reliability in **G-NAF Output** on page 522for information on reliability levels.

If a complete house number range/suffix is specified for input, candidates from a point data source must be fully matched. If partial house number information is given (without complete range or suffix information), then candidates with non-conflicting range/suffix information (or no range/suffix information) will match.

The following table shows geocoding match results with house number ranges from a point data source (G-NAF database.). The matching rules are based on the Address Matching Approval System (AMAS®) developed by Australia Post.

Table 71: Range Match Results

Input House Number	Data House Number	Candidate House Number	Match or Non-Match
10	10A	10A	Match: Input number 10 matches 10A (or 10 with any suffix).
10A	10	10	Match: Input number 10 matches.
10C	10A	10A	Non-Match: Input suffix does not match the data suffix.
10	10-12	10-12	Match: Input number 10 matches the first number of the dashed data range.
12	10-12	10-12	Match: Input number 12 matches the last number of the dashed data range.
10A	10-12	10-12	Match: Input number 10 matches the first number of the dashed data range. Input suffix is not in data, but this does not affect matching.
12	10-14	10-14	Non-Match. Input number 12 does not match either number of the dashed data range. No interpolation is performed on a house number ranges.
10-12	10-14	10-14	Match: First input number 10 matches the first number of the data and the second number 12 is within the data range.
10-12	10A-14A	10A-14A	Match: The input has no suffix information, but the input number 10 matches.

Input House Number	Data House Number	Candidate House Number	Match or Non-Match
10-16	10-12	10-12	Non Match: Input second number 16 is outside the 10-12 data range.
10-13	10-14	10-14	Non-Match: Both input numbers are within data range, but second number (13) is odd and this does not match with the even range of 10-14.
10-13	10-15	10-15	Match: Data range (10-15) suggests a mixed odd/even range, so input is matched.
RMB 10	10	10	Match: Input number matches
16	A16	A16	Match: Input number matches
RMB 10	A10	A10	Non Match: Input suffix does not match the data suffix.

Specifying Unit Information in an Address

When you use the G-NAF database, the geocoder recognizes several formats of unit types (such as unit, suite, floor, and flat address prefix) in an address, as illustrated below:

- "Unit 5 6 Macleay Street"—Full unit description used in conjunction with unit value and address number.
- "U 5 6 Macleay Street"—Abbreviated unit description used in conjunction with unit value and address number.
- "5/6 Macleay Street"—No Unit abbreviation with unit number and address number separated by a forward slash notation.

Exact matches on unit and address are placed first in the list of returned candidates.

Specifying Level Information in an Address

If you use the G-NAF database the geocoder can return level information for some addresses. Level information identifies the floor or level of a multi-storey building. The G-NAF database includes level information for some states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number).

If the G-NAF database has level information for an address, the geocoder returns that information with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information.

If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.

Following are several examples of partial addresses that contain level information. The level component is indicated in bold.

Suite 3 **Level 7**, 17 Jones Street (Suite 3 is a unit)

Floor 2, 17 Jones Street

Level 7, 17-19 Middleborough Road

Options

GeocodeAddressAUS allows you to set default processing options through the Management Console. You can override certain settings for individual calls to GeocodeAddressAUS using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

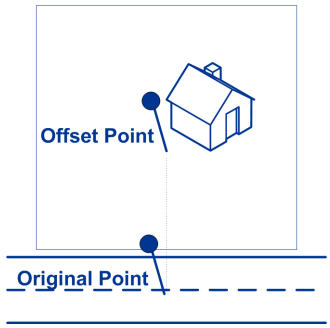
Geocoding Options

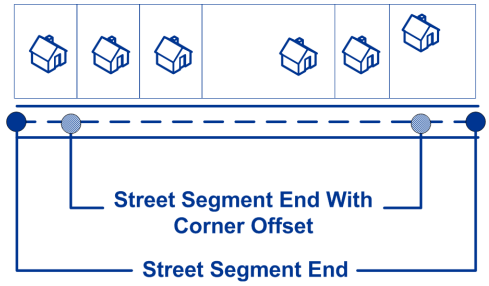
The following table lists the options that control how a location's coordinates are determined.

Table 72: Geocoding Options for Australia

optionName	Description / Valid Values	
GeocodeLevel	Specifies how precisely you want to geocode addresses. One of the following:	
	StreetAddress	The geocoder attempts to geocode addresses to a street address, but some matches may end up at a less precise location such as a postal code centroid, intersection, or shape path.
	PostalCenroid	The geocoder attempts to geocode addresses to the most precise postal code it finds. The advantage of postal code centroid matching is the speed of the operation. The disadvantage of postal code matching is that the geocoder only examines the PostalCode field. If you use street address precision, the geocoder looks at both the street name and the PostalCode field and attempts to return street-level coordinates and optionally fall back to postal code coordinates.
	GeographicCentroid	The geocoder attempts to geocode addresses to the geographic centroid of a city or state.

optionName	Description / Valid Values
Interpolation	Specifies whether to perform address point interpolation. This option only works if you have a point database, such as ParcelPrecision, installed. This option is not available for all countries. Address point interpolation uses point data to refine geocode results. By default, the geocoding process estimates the location of an address based on the street numbers at either end of street segment. For example, if a street segment runs from 100 Main St. to 200 Main St., then a request for 150 Main St. will return a location in the middle of the segment. With interpolation, the geocoder finds the position of 180 Main St. in the point data, and it is about two-thirds of the way down the street. Using this information, the geocoder can estimate the position of 150 Main St. based on 100 and 180 Main St. In this case, the geocoder estimates the location of the address slightly away from the center of the segment. One of the following: **Y** Yes, perform address point interpolation. **N** No, do not perform address point interpolation.
FallbackToGeographic	Specifies whether to attempt to determine a geographic region centroid when an address-level geocode cannot be determined. One of the following: **Y** Yes, determine a geographic centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a geographic centroid when an address-level centroid cannot be determined.
FallbackToPostal	Specifies whether to attempt to determine a postal code centroid when an address-level geocode cannot be determined. One of the following: **Y** Yes, determine a postal code centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a postal code centroid when an address-level centroid cannot be determined.
OffsetFromStreet	Indicates the offset distance from the street segments to use in street-level geocoding. The distance is specified in the units you specify in the OffsetUnits option.

optionName	Description / Valid Values
	The default is 10 meters. The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point.  Street coordinates are accurate to 1/10,000 of a degree and interpolated points are accurate to the millionths of a degree.
OffsetFromCorner	Specifies the distance to offset the street end points in street-level matching. The distance is specified in the units you specify in the OffsetUnits option.This value is used to prevent addresses at street corners from being given the same geocode as the intersection. The default is 12 meters. The following diagram compares the end points of a street to offset end points.

optionName	Description / Valid Values
	
OffsetUnits	Specifies the unit of measurement for the street offset and corner offset options. One of the following: • Feet • Miles • Meters • Kilometers The default is Meters.
GNAFPointType	For street address matching, specifies whether to return the parcel latitude/longitude or the street frontage latitude/longitude. This option is available only if you have the G-NAF database installed, and the database selected in the **Database** list on the Data tab includes the G-NAF database. This option only affects addresses matched to the G-NAF database. One of the following: **P** In a street address match, return the exact location of the parcel. This is the standard G-NAF point which is the exact authoritative point returned by the G-NAF database. Default. **S** In a street address match, return the street frontage point for the parcel. The street frontage point is 12.5 meters from the front boundary of the parcel. Street frontage points are more suitable for routing applications.
Return8DecimalPlaceParcelLatLong	Specifies whether to return the original latitude and longitude, precise up to eight digits after the decimal. This is the latitude/longitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This option is available only if you have the G-NAF database installed, and the database selected in the **Database** list on the Data tab includes the G-NAF database. This option only affects addresses matched to the G-NAF database.

optionName	Description / Valid Values
	One of the following: **Y** Yes, return the original latitude/longitude, up to eight digits after the decimal. **N** No, do not return the original latitude/longitude.
CoordinateSystem	A coordinate system is a reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry. Spectrum™ Technology Platform supports systems recognized by the European Petroleum Survey Group (EPSG). **EPSG:4283** Also known as the GDA94 coordinate system. **EPSG:4326** Also known as the WGS84 coordinate system. **EPSG:27200** Also known as the NZGD49 coordinate system.

Matching Options

Matching options let you set match restrictions, fallback, and multiple match settings so that the matching can be as strict or relaxed as you need. The strictest matching conditions require an exact match on house number, street name, postal code and no fallback to postal code centroids. The geocoder looks for an exact street address match within the postal code in the input address. Relaxing the conditions broadens the area in which it searches for a match. For example, by relaxing the postal code, the geocoder searches for candidates outside the postal code but within the city of your input address.

For guidelines on how to balance match rate and precision, see **Balancing Match Rate and Precision** on page 302.

Table 73: Matching Options for Australia

optionName	Description / Valid Values
KeepMultimatch	Specifies whether to return results when the address matches to multiple candidates in the database. If this option is not selected, an address that results in multiple candidates will fail to geocode. If you select this option, specify the maximum number of candidates to return using the MaxCandidates option (see below). One of the following: **Y** Yes, return candidates when multiple candidates are found. Default.

optionName	Description / Valid Values
	N No, do not return candidates. Addresses that result in multiple candidates will fail to geocode.
MaxCandidates	If you specify KeepMultimatch=Y, this option specifies the maximum number of results to return. The default is 1.
CloseMatchesOnly	Specifies whether to return only those geocoded results that are close match candidates. For example, if there are 10 candidates and two of them are close candidates, and you enable this option, only the two close matching candidates would be returned instead of all 10. To specify what is considered a close match, use the **MustMatch** options. Address candidates are ranked according to how closely the input address matches these preferences. One of the following: **Y** Yes, return only close matches. **N** No, do not return only close matches. Default.
MatchMode	Specifies how to determine whether a candidate is a close match. One of the following: **CustomMode** This option allows you to specify which parts of a candidate address must match the input address to be considered a close match. Use the **MustMatch<Element>** options to specify the address elements you want. This is the default value. **RelaxedMode** All candidates are considered close.
MustMatchInput	Specifies whether candidates must match all non-blank input fields to be considered a close match. For example, if an input address contains a city and postal code, then candidates for this address must match the city and postal code to be considered a close match. One of the following: **Y** Yes, a candidate must match all input to be considered a close match. **N** No, a candidate does not have to match all input to be considered a close match. Default.
MustMatchHouseNumber	Specifies whether candidates must match the house number to be considered a close match.

optionName	Description / Valid Values
	If you select this option you should also require an exact match on street name. This option does not significantly affect performance. It does, however, affect the type of match if the candidate address corresponds to a segment that does not contain any ranges. The type of match can also be affected when the house number range for a candidate does not contain the input house number. One of the following: **Y** Yes, a candidate must match the house number to be considered a close match. **N** No, a candidate does not have to match the house number to be considered a close match.
MustMatchStreet	Specifies whether candidates must match the street name to be considered a close match. If a close match is found, the geocoder attempts expanded street name manipulation, which looks for candidates with names that sound like the input address or that are spelled improperly. For example, "Muller Strasse" for "Mueller Strasse". This slows down performance but increases the match rate . If the geocoding database is indexed, the performance impact is reduced. One of the following: **Y** Yes, a candidate must match the street name to be considered a close match. **N** No, a candidate does not have to match the street name to be considered a close match.
MustMatchLocality	This option is not used for this country.
MustMatchCity	Specifies whether candidates must match the city to be considered a close match. If you do not require exact matches on city, the geocoder searches on the street address matched to the particular postal code, and considers other cities that do not match the name, but do match the postal code. One of the following: **Y** Yes, a candidate must match the city to be considered a close match. **N** No, a candidate does not have to match the city to be considered a close match.

optionName	Description / Valid Values
MustMatchCounty	Specifies whether candidates must match the Local Government Authority (LGA) to be considered a close match. One of the following: **Y** Yes, a candidate must match the county to be considered a close match. **N** No, a candidate does not have to match the county to be considered a close match.
MustMatchStateProvince	Specifies whether candidates must match the state to be considered a close match. One of the following: **Y** Yes, a candidate must match the state or province to be considered a close match. **N** No, a candidate does not have to match the state or province to be considered a close match.
MustMatchPostalCode	Specifies whether candidates must match the postal code to be considered a close match. If you do not require exact match on postal codes, the geocoder searches a wider area for a match. While this results in slower performance, the match rate is higher because the request does not need to match exactly when it compares match candidates. One of the following: **Y** Yes, a candidate must match the postal code to be considered a close match. **N** No, a candidate does not have to match the postal code to be considered a close match.
PreferPostalCodeOverCity	Specifies whether to return a close match when the postal code and street match exactly but the city does not. If you enable this option, the City input field is ignored if the postal code and street address are an exact match. If you do not enable this option, there may be situations where street address and postal code input are an exact match but the city name is not, and close matches are not returned. For example, without this option enabled, the following input address would return no close matches:

optionName	Description / Valid Values

	`5 East St`
	`GLENROY VIC 3046`
	
	With this option enabled, this same input address would return the following close match:
	
	`5 EAST ST`
	`HADFIELD VIC 3046`
	
	One of the following:
	
	Y Yes, return close matches if the postal code and street address are an exact match and ignore the city input.
	
	N No, do not return close matches if the postal code and street address are an exact match and the city is not. Default.

Data Options

The Data tab allows you to specify which databases to use in geocoding. Databases contain the address and geocode data necessary to determine the geocode for a given address. There are two kinds of databases: standard databases and custom databases. Standard databases are those supplied by Pitney Bowes Software and based on address and geocoding data from postal authorities and suppliers of geographical data. Custom databases are databases you create to enhance or augment standard databases for your particular needs. For more information on custom databases, see **What Is a Custom Database?** on page 602.

For Australian geocoding, to achieve the best geocoding spatial precision use the G-NAF database. This provides point-level geocoding that places points within the land parcel boundary for a given address. The G-NAF database requires an additional license. Contact your sales representative for more information.

The following table lists the options available for specifying which databases to use and the search order of databases.

Table 74: Data Options for Australia

optionName	Description / Valid Values
Database	Specifies the database to be used for geocoding. Only databases that have been defined in the Databases Resources panel in the Management Console are available.
DatabasePreference	Specifies which geocoding databases to use. One of the following:

optionName	Description / Valid Values
	PreferCustom Use both standard databases and custom databases, but give preference to candidates from custom databases. Use this option if you feel your custom database is superior to the standard database. **PreferStandard** Use both standard databases and custom databases, but give preference to candidates from standard databases. **CustomOnly** Use only custom databases. Ignore standard databases. **StandardOnly** Use only standard databases. Ignore custom databases. **Both** Use both standard databases and custom databases. In cases where candidates are returned from both, the standard database is preferred. Default. The results from a custom database have a "U" at the end of the result code. Results from an address database have an "A" at the end of the match score. For example: S5HPNTSCZA is a match score that comes from an address database, while S5HPNTSCZU comes from a custom database. For more information, see **Introduction** on page 597.
DatabaseSearchOrder	The name of one or more database resources to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. You can specify multiple database resources. If you specify more than one database, list them in order of preference. The order of the databases has an effect when there are close match candidates from different databases. The close matches that are returned come from the database that is first in the search list. Close matches from lower ranked databases are demoted to non-close matches. You can also use the order of the databases to perform fallback processing if you have an both an address point database and a street-level database installed for the country. List the address point database first and the street database second. If the address cannot be geocoded to the address point level, the geocoder will attempt to geocode it to the street level.

Output

GeocodeAddressAUS returns the latitude/longitude, standardized address, and result indicators. Result indicators describe how well the geocoder matched the input address to a known address and assigned a location; they also describe the overall status of a match attempt. The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section .

Address Output

The address may be identical to the input address if the input address was accurate, or it may be a standardized version of the input address, or it may be a candidate address when multiple matches are found.

Table 75: Address Output for Australia

columnName	Description
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
City	The municipality name.
Country	The three-letter ISO 3166-1 Alpha 3 country code. For Australia, the country code is AUS.
County	The Local Government Authority (LGA) name.
FirmName	Name of the company or a place name.
HouseNumber	The building number for the matched location.
HouseNumberHigh	The highest house number of the range in which the address resides.
HouseNumberLow	The lowest house number of the range in which the address resides.
HouseNumberParity	Indicates if the house number range contains even or odd numbers or both. **E** Even

columnName	Description
	O Odd **B** Both **U** Unknown
LastLine	Complete last address line (city, state/province, and postal code).
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
NumberOfCandidateRanges	Indicates whether the address has a house number. One of the following: **0** The address has no house number. Examples of addresses that have no house number are P.O. box addresses and general delivery addresses. **1** The address has a house number. For information about the range in which the house number falls, see the HouseNumberHigh, HouseNumberLow, and HouseNumberParity fields.
NumberOfRangeUnits	Indicates whether or not the address has a unit number, such as a suite number or apartment number. One of the following: **0** The address has no unit number. **1** The address has a unit number. For information about the range in which the unit number falls, see the UnitNumberHigh and UnitNumberLow.
PostalCode	The postcode for the address. The format of the postcode varies by country.
PostalCode.Addon	The second part of a postcode. For example, for Canadian addresses this will be the LDU. This field is not used by most countries.
PreAddress	Miscellaneous information that appears before the street name.
PrivateMailbox	This field is not currently used.
SegmentCode	A unique ID that identifies a street segment.
SegmentParity	Indicates which side of the street has odd numbers.

columnName	Description
	L Left side of the street **R** Right side of the street **B** Both sides of the street **U** Undetermined
StateProvince	The state name.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.
StreetName	The street name.
StreetPrefix	The type of street when the street type appears before the base street name. For example, AVENUE: 12 **AVENUE** B KALGOORLIE WA 6430
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 Washington N.
UnitNumberHigh	The highest unit number of the range in which the unit resides.
UnitNumberLow	The lowest unit number of the range in which the unit resides.

Geocode Output

Table 76: Geocode Output for Australia

columnName	Description
CoordinateSystem	The coordinate system used to determine the latitude and longitude coordinates. A coordinate system specifies a map projection, coordinate units, etc. An example is EPSG:4326. EPSG stands for European Petroleum Survey Group.

columnName	Description
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

Result Codes

Result codes contain information about the success or failure of the geocoding attempt, as well as information about the accuracy of the geocode.

Table 77: Result Code Output for Australia

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. For more information, see **Introduction** on page 597.
IsCloseMatch	Indicates whether or not the address is considered a close match. An address is considered close based on the "Close match criteria" options on the Matching tab. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
MultiMatchCount	For street address geocoding, the number of matching address positions found for the specified address. For intersection geocoding, the number of matching street intersection positions found for the specified addresses.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	If the geocoder could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data

columnName	Description
	• Multiple Matches Found • Exception occurred • Unable to initialize Geocoder • No Match Found
Status.Description	If the geocoder could not process the address, this field will show a description of the failure. **Problem + explanation** — Returned when Status.Code = Internal System Error. **Geocoding Failed** — Returned when Status.code = No Geocode Found. **No location returned** — Returned when Status.code = No Geocode Found. **No Candidates Returned** — The geocoder could not identify any candidate matches for the address. **Multiple Candidates Returned and Keep Multiple Matches not selected** — The address resulted in multiple candidates. In order for the candidate address to be returned, you must specify `KeepMultimatch=Y`.
LocationPrecision	A code describing the precision of the geocode. One of the following: **0** — No coordinate information is available for this candidate address. **1** — Interpolated street address. **2** — Street segment midpoint. **3** — Postal code 1 centroid. **4** — Partial postal code 2 centroid. **5** — Postal code 2 centroid. **6** — Intersection. **7** — Point of interest. **8** — State/province centroid. **9** — County centroid. **10** — City centroid. **11** — Locality centroid.

columnName	Description	
	12	Additional point precision for unspecified custom item.
	13	Additional point precision for unspecified custom item.
	14	Additional point precision for unspecified custom item.
	15	Additional point precision for unspecified custom item.
	16	The result is an Address Point.
	17	The result was generated by using address point data to modify the candidates segment data.

Australia G-NAF Database Output

The following table lists output fields that are unique to the Australian Geocoded National Address File (G-NAF®) database. G-NAF is an optional database that is available for all six states and two territories. G-NAF is the only authoritative Australian national index of locality, street and number, validated with geographic coordinates.

Table 78: Austraila G-NAF Output

columnName	Description	
AUS.GNAF_CONFIDENCE	A number indicating how many G-NAF datasets the address is found in. A higher confidence level means that the same address was found in more data contributor sources. One of the following:	
	<number>	The number of datasets the address was found in, minus 1. For example, a value of 0 indicates that the address was found in one contributor's dataset, a value of 1 indicates that the address was found in two contributors' datasets, a value of 2 indicates that the address was found in three contributors' datasets, and so forth.
	-1	The address could not be found in any G-NAF dataset.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LATITUDE	The parcel latitude, precise to eight digits after the decimal. This is the latitude that the candidate matched to in the G-NAF database. These are the	

columnName	Description
	original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LONGITUDE	The parcel longitude, precise to eight digits after the decimal. This is the longitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_GEOCODE_LEVEL	A number indicating the level o f geocode for the address. Every principal address within the G-NAF database has at least a locality level geocode. They may also have a street level geocode and a point level geocode. One of the following: **0** No geocode. **1** Parcel level geocode only (no locality or street level geocode). **2** Street level geocode only (no locality or parcel level geocode). **3** Street and parcel level geocodes (no locality geocode). **4** Locality level geocode only (no street or parcel level geocode). **5** Locality and parcel level geocodes (no street level geocode). **6** Locality and street level geocodes (no parcel level geocodes). **7** Locality, street and parcel level geocodes.
AUS.GNAF_PID	The G-NAF Persistent Identifier (G-NAF PID) is a 14-character alphanumeric string that uniquely identifies each G-NAF address. The PID is constructed from a combination of the major

columnName	Description
	address fields of the G-NAF Dictionary. An example of a G-NAF PID is: GAVIC411711441
AUS.GNAF_RELIABILITY	A number indicating the geocode precision. Reliability is related to the dictionary used to determine the geocode. Data with geocoded reliability levels 1, 2, and 3 is contained in the GNAF123 Dictionary. This is point (parcel) level geocoded data. Data with geocoded reliability levels 4, 5, and 6 is contained in the GNAF456 Dictionary. This contains non-parcel centroid geocoded data. **1** Geocode accuracy recorded to appropriate surveying standard. For example, this could apply to an address level geocode that was manually geocoded. Geocode resolution is sufficient to place the centroid within address site boundary with a GPS. **2** Geocode accuracy sufficient to place centroid within address site boundary. For example, this could apply to an address level geocode that was automatically calculated as the centroid of the corresponding cadastre parcel. **3** Geocode accuracy sufficient to place centroid near (or possibly within) address site boundary. For example, this could apply to an address level geocode that was automatically calculated by calculating where on the road the address was likely to appear based upon other bounding geocoded addresses. **4** Geocode accuracy sufficient to associate address site with a unique road feature. For example, this could apply to a street level geocode that was automatically calculated by using the road centerline reference data. **5** Geocode resolution sufficient to associate address site with a unique locality or neighborhood. For example, this could apply to a locality level geocode that was automatically calculated as the centroid of the locality.

columnName	Description
	6 Geocode resolution sufficient to associate address site with a unique region. For example, this could apply to a locality level geocode that was derived from topographic feature.
AUS.LEVEL_NUMBER	The number of a floor or level in a multi-story building. For example, **Floor 2** , 17 Jones Street The G-NAF database includes level information for some Australian states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.LEVEL_TYPE	The label used for a floor of a multi-story building. For example, "Level" or "Floor". In this example, the level type is "Level": Suite 3 **Level 7**, 17 Jones Street In this example, Suite 3 is a unit. The G-NAF database includes level information for some Australian states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an

columnName	Description
	address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.MESH_BLOCK_ID	A Meshblock is the smallest geographic unit for which statistical data is collected by the Australian Bureau of Statistics (ABS). Meshblocks usually contain a minimum of 20 to 50 households. This is about one fifth the size of a Collection District (CD). You can use the Meshblock ID to do additional attributions against your own data.

GeocodeAddressGBR

GeocodeAddressGBR takes an address in the United Kingdom and returns the geographic coordinates. GeocodeAddressGBR can geocode to street addresses and postal code centroids.

GeocodeAddressGBR is an optional part of the Enterprise Geocoding Module. For more information about Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Input

Input Fields

GeocodeAddressGBR takes an address. To obtain the best performance and the most possible matches, your input address lists should be as complete as possible, free of misspellings and incomplete addresses, and as close to postal authority standards as possible. Most postal authorities have websites that contain information about address standards for their particular country.

The following table lists the input fields used for geocoding locations in the United Kingdom.

Note: Specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 79: Input Fields for Great Britain

columnName	Description / Valid Values
AddressLine1	One of the following: • The address line containing the street name and building number. This can contain a building name. For example: • The full address. For more information, see **Single Line Input** on page 335
AddressLine2	The second address line of a two-line address. For example: Minton Place **Victoria Street** Windsor SL4 1EG
City	The city or town name. Your input address should use the official city name.
County	This field is not used in this country.
FirmName	Company name. For example: **Pitney Bowes Ltd** Elizabeth Way Harlow CM19 5BD
HouseNumber	The building number. You may get better parsing results for some countries if you put the house number in this field instead of AddressLine1. For the best results, use the HouseNumber input field for these countries: India, Singapore, Malaysia, Thailand, and Turkey when using multi-line input. **Note:** The house number specified in the HouseNumber field takes precedence over any house number specified in the AddressLine1 field.
LastLine	The last line of the address. Minton Place Victoria Street **Windsor SL4 1EG**
Locality	The locality name.

columnName	Description / Valid Values
PostalCode	The postal code in the appropriate format for the country. In the United Kingdom, each address is associated with an alphanumeric code up to seven characters in length. Each postcode includes an average of 15 addresses. In some cases, where a customer receives a substantial amount of mail, for example, a business, the postcode pertains to just that one address (a large-user postcode).
StateProvince	The region name.

Address Guidelines for Great Britain

Follow these suggestions to ensure that your street input data is in the best format possible for optimum geocoding.

- **Street addresses**—If you provide a street address, and either a town and state, or a postcode, you can perform an address geocode. The geocoder will match your full address record against its comprehensive geocoding database. Minor misspellings in street addresses are corrected in the returned candidates.
- **Post Office Box numbers**—You can enter PO Box input in AddressLine1 in the format: [PO Box or Post Box] [number]. The number typically ranges from one digit to a maximum of six digits. The acceptable formats for PO Box include the following. These formats are not case sensitive.
  - PO BOX
  - P.O. BOX
  - P.O.BOX
  - PO.BOX
  - POST BOX
  - POST BOX Number
  - POST BOX No
  - POST BOX #
  - POST OFFICE BOX
  - POBOX
  - POB
  - POSTBOX
  - P.BOX
  - PBOX
  - P BOX
  - BOX
- **Locality**—UK addresses may contain two localities – dependent locality and double dependent locality, both of which you can enter in the **Locality** input field:

- **Dependent Locality**—Distinguishes an address when there are identically named thoroughfares within the same postal town. For example, **Winford**, Bristol.
- **Double Dependent Locality**—Distinguishes addresses between similar or identically named thoroughfares within the same locality. This can typically be a business park, estate park or airport. For example, Brunel Way, **Cranford Estate**, Millbrook, Southampton.

A locality alias is not part of the postal address but can be used to help find addresses were the town or postcode is unknown. For example, Rycote Lane, **Rycote**.

- **Postal code**—If you enter the postcode in the **LastLine** input field, you may use either the full postcode or just the post district as long as other street information or post information is present. If you enter the postcode in the **PostalCode** input field, you must use the complete postcode. Providing both the town and postcode can produce better results. The geocoder corrects postcode information and adds postcodes if your input does not already include them.
- **Northern Ireland addresses**—The AddressPoint and CodePoint databases include the same Northern Ireland data, accurate to the postcode centroid level. The AddressPoint source data source does not include Northern Ireland addresses, however the AddressPoint dictionary is augmented with Royal Mail postcode data for Northern Ireland. This Northern Ireland data has postcode centroid (result code S3) precision only. Therefore, Northern Ireland addresses are geocoded to the same precision level with AddressPoint or CodePoint dictionaries. For Northern Ireland addresses, an S3 result code is the highest level of precision, even with the AddressPoint dictionary.

Single Line Input

Instead of entering each address element in separate fields, you may enter the entire address in the AddressLine1 input field.

You can enter addresses in this single-line format.

```
StreetAddress;Locality;City;PostalCode
```

Note: U.K. addresses must be in this format. You must include either a city plus state/province or a postal code as the last address element.

Where:

- *StreetAddress* can be house number and street name in either order (with street type immediately before or after the street name).
- *City* is the town.
- *Locality* is the locality name.
- *PostalCode* is the complete postcode.

The matching accuracy for single line input is comparable to that of structured address input. The performance of single line input addresses may be slightly slower than that of structured address input.

For best results, use delimiters (comma, semicolon, or colon) between each component of the address. For example,

```
Flat 69 Victoria Street,Windsor,SL4 1EH
```

If the input address is missing delimiters, spaces are recognized as separators and internal parsing rules identify address components. In the example above, the address would still successfully geocode even if some or all of the delimiters were missing in the input.

Note: Non-delimited or partially-delimited single line addresses may take longer to geocode and may not produce the same results as delimited single line input. This is especially true for addresses with multi-word street names or cities. To optimize single line geocoding, use delimiters between address components (particularly between street name and city).

Punctuation is ignored for geocoding purposes.

Guidelines for Single Line Input

- The country is not required. Each country geocoder assumes that the address is in its country.
- Firm information (placename, building name, or government building) is returned if available.

Options

allows you to set default processing options through the Management Console. You can override certain settings for individual calls to using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

Geocoding Options

The following table lists the options that control how a location's coordinates are determined.

Table 80: Geocoding Options for Great Britain

optionName	Description / Valid Values	
GeocodeLevel	Specifies how precisely you want to geocode addresses. One of the following:	
	StreetAddress	The geocoder attempts to geocode addresses to a street address, but some matches may end up at a less precise location such as a postal code centroid, intersection, or shape path.
	PostalCenroid	The geocoder attempts to geocode addresses to the most precise postal code it finds. The advantage of postal code centroid matching is the speed of the operation. The disadvantage of postal code matching is that the geocoder only examines the PostalCode field. If you use street address precision, the geocoder looks at both the street name and the PostalCode field and attempts to return street-level coordinates and optionally fall back to postal code coordinates.
	GeographicCentroid	The geocoder attempts to geocode addresses to the geographic centroid of a city or state. This

optionName	Description / Valid Values
	option is not available for the United Kingdom (GBR).
FallbackToGeographic	Specifies whether to attempt to determine a geographic region centroid when an address-level geocode cannot be determined. This option is not available for the United Kingdom (GBR). One of the following: **Y** Yes, determine a geographic centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a geographic centroid when an address-level centroid cannot be determined.
FallbackToPostal	Specifies whether to attempt to determine a postal code centroid when an address-level geocode cannot be determined. One of the following: **Y** Yes, determine a postal code centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a postal code centroid when an address-level centroid cannot be determined.
CoordinateSystem	A coordinate system is a reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry. Spectrum™ Technology Platform supports systems recognized by the European Petroleum Survey Group (EPSG). **EPSG:4326** Also known as the WGS84 coordinate system. **EPSG:27200** Also known as the NZGD49 coordinate system. **EPSG:27700** Also known as the British National Grid system.

Matching Options

Matching options let you set match restrictions, fallback, and multiple match settings so that the matching can be as strict or relaxed as you need. The strictest matching conditions require an exact match on house number, street name, postal code and no fallback to postal code centroids. The geocoder looks for an exact street address match within the postal code in the input address. Relaxing the conditions

broadens the area in which it searches for a match. For example, by relaxing the postal code, the geocoder searches for candidates outside the postal code but within the city of your input address.

For guidelines on how to balance match rate and precision, see **Balancing Match Rate and Precision** on page 302.

Table 81: Matching Options for Great Britain

optionName	Description / Valid Values
KeepMultimatch	Specifies whether to return results when the address matches to multiple candidates in the database. If this option is not selected, an address that results in multiple candidates will fail to geocode. If you select this option, specify the maximum number of candidates to return using the MaxCandidates option (see below). One of the following: **Y** Yes, return candidates when multiple candidates are found. Default. **N** No, do not return candidates. Addresses that result in multiple candidates will fail to geocode.
MaxCandidates	If you specify KeepMultimatch=Y, this option specifies the maximum number of results to return. The default is 1.
ExpandCandidates	Specifies whether to return multiple close matches (one for each delivery point on the street). By default, if a street has multiple addresses/delivery points, GeocodeAddressGBR returns a single close match, with each individual delivery point available as a range.
CloseMatchesOnly	Specifies whether to return only those geocoded results that are close match candidates. For example, if there are 10 candidates and two of them are close candidates, and you enable this option, only the two close matching candidates would be returned instead of all 10. To specify what is considered a close match, use the **MustMatch** options. Address candidates are ranked according to how closely the input address matches these preferences. One of the following: **Y** Yes, return only close matches. **N** No, do not return only close matches. Default.
MatchMode	Specifies how to determine whether a candidate is a close match. One of the following:

optionName	Description / Valid Values
	CustomMode This option allows you to specify which parts of a candidate address must match the input address to be considered a close match. Use the **MustMatch<Element>** options to specify the address elements you want. This is the default value. **RelaxedMode** All candidates are considered close.
MustMatchInput	Specifies whether candidates must match all non-blank input fields to be considered a close match. For example, if an input address contains a city and postal code, then candidates for this address must match the city and postal code to be considered a close match. One of the following: **Y** Yes, a candidate must match all input to be considered a close match. **N** No, a candidate does not have to match all input to be considered a close match. Default.
MustMatchHouseNumber	Specifies whether candidates must match the house number to be considered a close match. If you select this option you should also require an exact match on street name. This option does not significantly affect performance. It does, however, affect the type of match if the candidate address corresponds to a segment that does not contain any ranges. The type of match can also be affected when the house number range for a candidate does not contain the input house number. One of the following: **Y** Yes, a candidate must match the house number to be considered a close match. **N** No, a candidate does not have to match the house number to be considered a close match.
MustMatchStreet	Specifies whether candidates must match the street name to be considered a close match. If a close match is found, the geocoder attempts expanded street name manipulation, which looks for candidates with names that sound like the input address or that are spelled improperly. For example, "Muller Strasse" for "Mueller Strasse". This slows down performance but increases the match rate . If the geocoding database is indexed, the performance impact is reduced.

optionName	Description / Valid Values
	One of the following: **Y** Yes, a candidate must match the street name to be considered a close match. **N** No, a candidate does not have to match the street name to be considered a close match.
MustMatchLocality	Specifies whether candidates must match the locality to be considered a close match. If you do not require exact matches on locality, the geocoder searches on the street address matched to the particular postal code, and considers other localities that do not match the name, but do match the postal code. This option is not available if you specify **GeocodeLevel=GeographicCentroid**. Specify one of the following values: **Y** Yes, a candidate must match the locality to be considered a close match. **N** No, a candidate does not have to match the locality to be considered a close match.
MustMatchCity	Specifies whether candidates must match the city to be considered a close match. If you do not require exact matches on city, the geocoder searches on the street address matched to the particular postal code, and considers other cities that do not match the name, but do match the postal code. One of the following: **Y** Yes, a candidate must match the city to be considered a close match. **N** No, a candidate does not have to match the city to be considered a close match.
MustMatchCounty	Specifies whether candidates must match the county to be considered a close match. One of the following: **Y** Yes, a candidate must match the county to be considered a close match.

optionName	Description / Valid Values
	N No, a candidate does not have to match the county to be considered a close match.
MustMatchStateProvince	Specifies whether candidates must match the region to be considered a close match. One of the following: **Y** Yes, a candidate must match the state or province to be considered a close match. **N** No, a candidate does not have to match the state or province to be considered a close match.
MustMatchPostalCode	Specifies whether candidates must match the postal code to be considered a close match. If you do not require exact match on postal codes, the geocoder searches a wider area for a match. While this results in slower performance, the match rate is higher because the request does not need to match exactly when it compares match candidates. One of the following: **Y** Yes, a candidate must match the postal code to be considered a close match. **N** No, a candidate does not have to match the postal code to be considered a close match.
MustMatchPostalDistrict	Specifies whether the postal district portion of the postcode must match in order for the match to be considered a close match. UK postcodes are divided into two sections: the outward code, which is to the left of the space, and the inward code, which is to the right. The outward code represents the postal district. For example, in the postcode CB3 OHH, the postal district is CB3, which is Cambridge.

Data Options

The Data tab allows you to specify which databases to use in geocoding. Databases contain the address and geocode data necessary to determine the geocode for a given address. There are two kinds of databases: standard databases and custom databases. Standard databases are those supplied by Pitney Bowes Software and based on address and geocoding data from postal authorities and suppliers of geographical data. Custom databases are databases you create to enhance or augment standard databases for your particular needs. For more information on custom databases, see **What Is a Custom Database?** on page 602.

The following table lists the options available for specifying which databases to use and the search order of databases.

Table 82: Data Options for Great Britain

optionName	Description / Valid Values
Database	Specifies the database to be used for geocoding. Only databases that have been defined in the Databases Resources panel in the Management Console are available.
DatabasePreference	Specifies which geocoding databases to use. One of the following: **PreferCustom** Use both standard databases and custom databases, but give preference to candidates from custom databases. Use this option if you feel your custom database is superior to the standard database. **PreferStandard** Use both standard databases and custom databases, but give preference to candidates from standard databases. **CustomOnly** Use only custom databases. Ignore standard databases. **StandardOnly** Use only standard databases. Ignore custom databases. **Both** Use both standard databases and custom databases. In cases where candidates are returned from both, the standard database is preferred. Default. **Note:** This option is not available for Great Britain (GBR) because the GBR geocoder does not support custom databases. The results from a custom database have a "U" at the end of the result code. Results from an address database have an "A" at the end of the match score. For example: S5HPNTSCZA is a match score that comes from an address database, while S5HPNTSCZU comes from a custom database. For more information, see **Introduction** on page 597.
DatabaseSearchOrder	The name of one or more database resources to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. You can specify multiple database resources. If you specify more than one database, list them in order of preference.

optionName	Description / Valid Values
	The order of the databases has an effect when there are close match candidates from different databases. The close matches that are returned come from the database that is first in the search list. Close matches from lower ranked databases are demoted to non-close matches. You can also use the order of the databases to perform fallback processing if you have an both an address point database and a street-level database installed for the country. List the address point database first and the street database second. If the address cannot be geocoded to the address point level, the geocoder will attempt to geocode it to the street level.

Output Data Options

The following table lists the options that control which data is returned in the output.

Table 83: Output Data Options

optionName	Description / Valid Values
ReturnOnlySimilarFirmNames	Specifies whether to return firm names only when the input firm name is similar to the firm name in the geocoding database. For example, if the input firm name is "Pitney Bowes Business Insight" but the geocoding database returns "Pitney Bowes Software, Inc.", these two firm names are not similar. In most cases the input firm name must match the firm name in the database exactly. Some differences in abbreviations are considered similar enough to result in the firm name being returned. One of the following: **Y** Yes, return only firm names that are similar to the input firm name. **N** No, return firm names regardless of whether they are close to the input firm name. Default.

Output

GeocodeAddressGBR returns the latitude/longitude, standardized address, and result indicators. Result indicators describe how well the geocoder matched the input address to a known address and assigned a location; they also describe the overall status of a match attempt. The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section .

Address Output

The address may be identical to the input address if the input address was accurate, or it may be a standardized version of the input address, or it may be a candidate address when multiple matches are found.

Table 84: Address Output for Great Britain

columnName	Description
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
City	The municipality name.
Country	The three-letter ISO 3166-1 Alpha 3 country code. For the United Kingdom, the country code is GBR.
County	This field is not used in this country.
FirmName	Name of the company or a place name.
HouseNumber	The building number for the matched location.
HouseNumberHigh	The highest house number of the range in which the address resides.
HouseNumberLow	The lowest house number of the range in which the address resides.
HouseNumberParity	Indicates if the house number range contains even or odd numbers or both. **E** Even **O** Odd **B** Both **U** Unknown
LastLine	Complete last address line (city, state/province, and postal code).

columnName	Description
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
Locality	The locality name.
NumberOfCandidateRanges	Indicates whether the address has a house number. One of the following: **0** The address has no house number. Examples of addresses that have no house number are P.O. box addresses and general delivery addresses. **1** The address has a house number. For information about the range in which the house number falls, see the HouseNumberHigh, HouseNumberLow, and HouseNumberParity fields.
NumberOfRangeUnits	Indicates whether or not the address has a unit number, such as a suite number or apartment number. One of the following: **0** The address has no unit number. **1** The address has a unit number. For information about the range in which the unit number falls, see the UnitNumberHigh and UnitNumberLow.
PostalCode	The postcode for the address. The format of the postcode varies by country.
PostalCode.Addon	The second part of a postcode. For example, for Canadian addresses this will be the LDU. This field is not used by most countries.
PreAddress	Miscellaneous information that appears before the street name.
PrivateMailbox	This field is not currently used.
SegmentCode	A unique ID that identifies a street segment.
SegmentParity	Indicates which side of the street has odd numbers. **L** Left side of the street **R** Right side of the street **B** Both sides of the street

columnName	Description
	U Undetermined
StateProvince	The region name.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.
StreetName	The street name.
StreetPrefix	The type of street when the street type appears before the base street name. For example, AVENUE: 12 **AVENUE** B KALGOORLIE WA 6430
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 Washington N.
UnitNumberHigh	The highest unit number of the range in which the unit resides.
UnitNumberLow	The lowest unit number of the range in which the unit resides.

Geocode Output

Table 85: Geocode Output for Great Britain

columnName	Description
CoordinateSystem	The coordinate system used to determine the latitude and longitude coordinates. A coordinate system specifies a map projection, coordinate units, etc. An example is EPSG:4326. EPSG stands for European Petroleum Survey Group.
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

columnName	Description
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

Result Codes

Result codes contain information about the success or failure of the geocoding attempt, as well as information about the accuracy of the geocode.

Table 86: Result Code Output for Great Britain

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. For more information, see **Introduction** on page 597.
IsCloseMatch	Indicates whether or not the address is considered a close match. An address is considered close based on the "Close match criteria" options on the Matching tab. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
MultiMatchCount	For street address geocoding, the number of matching address positions found for the specified address. For intersection geocoding, the number of matching street intersection positions found for the specified addresses.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	If the geocoder could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data • Multiple Matches Found • Exception occurred • Unable to initialize Geocoder

columnName	Description
	• No Match Found
Status.Description	If the geocoder could not process the address, this field will show a description of the failure. **Problem + explanation** — Returned when Status.Code = Internal System Error. **Geocoding Failed** — Returned when Status.code = No Geocode Found. **No location returned** — Returned when Status.code = No Geocode Found. **No Candidates Returned** — The geocoder could not identify any candidate matches for the address. **Multiple Candidates Returned and Keep Multiple Matches not selected** — The address resulted in multiple candidates. In order for the candidate address to be returned, you must specify `KeepMultimatch=Y`.
LocationPrecision	A code describing the precision of the geocode. One of the following: **0** — No coordinate information is available for this candidate address. **1** — Interpolated street address. **2** — Street segment midpoint. **3** — Postal code 1 centroid. **4** — Partial postal code 2 centroid. **5** — Postal code 2 centroid. **6** — Intersection. **7** — Point of interest. **8** — State/province centroid. **9** — County centroid. **10** — City centroid. **11** — Locality centroid. **12** — Additional point precision for unspecified custom item. **13** — Additional point precision for unspecified custom item. **14** — Additional point precision for unspecified custom item.

columnName	Description	
	15	Additional point precision for unspecified custom item.
	16	The result is an Address Point.
	17	The result was generated by using address point data to modify the candidates segment data.

Country-Specific Output

The following table lists output fields that are unique to the United Kingdom.

Table 87: United Kingdom Output

columnName	Description
GBR.AP_STATUS_FLAG	AddressPoint status flag. This field is only returned if you are using the Address Point database. This status flag defines the quality and accuracy of each address in the Address Point database. For example, 0354. For more information, see the **Address Point User Guide** available at **www.ordnancesurvey.co.uk**.
GBR.DEPENDENT_STREET_NAME	Addresses in the United Kingdom may contain two street names: a main street name and dependent street name. Some addresses may not contain a street name at all.
GBR.DEPENDENT_LOCALITY	Dependent locality name. A dependent locality is a large village or district, for example, Wimbledon.
GBR.DOUBLE_DEPENDENT_LOCALITY	Double dependent locality name. A double dependnet locality is a small village or subdistrict.
GBR.HISTORIC_POSTCODE	If the input address contained an old postal code that has been replaced by a new postal code, this field contains the old postal code.
GBR.ALIASED_LOCALITY	A locality that is not part of the postal address.
GBR.OSAPR	Ordnance Survey AddressPoint Reference (OSAPR). Each address from the Address Point database has a unique OSAPR. OSAPRs are

columnName	Description
	always 18 characters long and start with the letters AP. For example, APMM918D7LQ65VG005.

GeocodeAddressGlobal

GeocodeAddressGlobal provides street-level geocoding for many countries. It can also determine city or locality centroids, as well as postal code centroids. GeocodeAddressGlobal handles street addresses in the native language and format. For example, a typical French formatted address might have a street name of Rue des Remparts. A typical German formatted address could have a street name Bahnhofstrasse.

Note: GeocodeAddressGlobal does not support U.S. or U.K. addresses. To geocode U.S. addresses, use GeocodeUSAddress. To geocode U.K. addresses, use GeocodeAddressGBR.

The countries available to you depends on which country databases you have installed. For example, if you have databases for Canada, Italy, and Australia installed, GeocodeAddressGlobal would be able to geocode addresses in these countries in a single stage. Before you can work with GeocodeAddressGlobal, you must define a global database resource containing a database for one or more countries. Once you create the database resource, a GeocodeAddressGlobal will become available in the Management Console, Enterprise Designer, and Interactive Driver. .For information on creating a Geocode Address Global database resource, see the *Spectrum™ Technology Platform User's Guide*.

GeocodeAddressGlobal is an optional component of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, see **What is the Enterprise Geocoding Module?** on page 294.

Input

GeocodeAddressGlobal takes an address or intersection as input. To obtain the best performance and the most possible matches, your input address lists should be as complete as possible, free of misspellings and incomplete addresses, and as close to postal authority standards as possible. Most postal authorities have websites that contain information about address standards for their particular country.

Input Fields

To obtain the best performance and the most possible matches, your input address lists should be as complete as possible, free of misspellings and incomplete addresses, and as close to postal authority standards as possible. Most postal authorities have websites that contain information about address standards for their particular country.

The following table lists the input fields used for geocoding.

Note: Specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 88: Input Fields

columnName	Description / Valid Values
AddressLine1	One of the following: • The address line containing the street name and building number.For Japanese addresses, the block and lot number. For addresses in the U.K., this can contain a building name. For example: **Muldenweg 2** 71665 Vaihingen an der Enz **9, rue Paul Lafargue** 93217 ST DENIS CEDEX ３番２２号 **calle naranjo 7** 89600 Altamira TAMPS • The full address. For more information, see **Single Line Input** on page 388 • For all countries except Argentina, Great Britain, and Japan, a street intersection. To specify a street intersection, use double ampersand (&&) to separate the streets. For more information, see **Street Intersection Input** on page 390. For France, an input street address can include a numbered range. For example, consider an input address of 104-106 rue de Charenton. The returned candidate includes two address ranges, and the 104 close match is from the 100-106 range. Alphanumeric ranged addresses are also handled (for example, you could input a alphanumeric ranged address like 2A-4B. If the geocoding database has alphabetic values for the input house number, the geocoder returns the house number as it exists in the database (with or without the alphabetic character). If the geocoder cannot confirm alphabetic values for the input house number, it returns the alphabetic value that was provided on input (as long as the house number matched).
AddressLine2	The second address line of a two-line address. For example: 26 WELLINGTON ST E **SUITE 500** TORONTO ON M5E 1S2 This field is not used in Argentina, Australia, Austria, Belgium or Luxembourg, Brazil, Chile, Czech Republic, Denmark, Estonia, Finland,

columnName	Description / Valid Values
	France, Germany, Hungary, India, Indonesia, Ireland, Italy, Japan, Latvia, Lithuania, Malaysia, Mexico, The Netherlands, Norway, Poland, Portugal, Singapore, Slovenia, South Africa, Spain, Sweden, Switzerland (including Liechtenstein), Thailand, Taiwan, and Turkey.
City	The city or town name. Your input address should use the official city name. For Argentina, Austria, the Czech Republic, Indonesia, Italy, Mexico, Portugal, Spain, Slovenia, and Switzerland, you may use the town alias. **Note:** In Argentina, Buenos Aires Federal District is not part of Buenos Aires province. If your input specifies only "Buenos Aires", Spectrum™ Technology Platform returns candidates in both the Federal District and in the region of Buenos Aires. For provincial capitals in Argentina, you can use the word Capital as well as the actual capital name. For example, input of "Capital, MZA" is equivalent to "Mendoza, MZA". Some areas in France are generally recognized as cities even though they are not truly administrative cities. These areas represent Artificial City Areas, or Virtual Towns. For a listing of supported virtual towns, see **Address Guidelines for France** on page 367. For Indonesia, this field contains the subdistrict. For example, Jakarta is an alias for Jakarta Utara. For Thailand, this field contains the subdistrict (tambon). For Japan, this field contains the municipality subdivision (oaza). For Lithuania, the city and county can be swapped on input. That is, if a city appears in either the City input field or in the County input field, the geocoder will still return the correct candidate. Similarly, a locality can be entered in either the Locality or City input field.
County	The meaning of county varies by country: • ARG (Argentina)—Department • AUS (Australia)—The Local Government Authority (LGA) • AUT (Austria)—Not used • BEL (Belgium)—Province • BRA (Brazil)—Not used • CAN (Canada)—Not used • CHE (Switzerland)—Province • CHL (Chile)—District • CHN (China)—Not used • COL (Columbia)—TBD

columnName	Description / Valid Values
	• CZE (Czech Republic)—District name or alias • DEU (Germany)—Kreis • DNK (Denmark)—Province • ESP (Spain)—Province • EST (Estonia)—District • FIN (Finland)—Not used • FRA (France)—Department • GBR (Great Britain)—Not used • GRC (Greece) —District • HKG (Hong Kong)—Not used • HUN (Hungary)—District • IDN (Indonesia)—Regency/City • IND (India)—District • IRL (Ireland)—Province • ITA (Italy)—Province • JPN (Japan)—City (shi) • LTU (Lithuania)—County • LVA (Latvia)—District • MAC (Macau)—Not used • MEX (Mexico)—Province • MYS (Malaysia)—Not used • NLD (The Netherlands)—Province • NOR (Norway)—District (fylke/counties) • NZL (New Zealand)—Region • PHL (Philippines)—District • POL (Poland)—District (poviat) • PRT (Portugal)—Not used • RUS (Russia) —District • SGP (Singapore)—Not used • SVK (Slovakia)—Town • SVN (Slovenia)—Region • SWE (Sweden)—Kommun • THA (Thailand)—District (amphoe) • TUR (Turkey)—District • TWN (Taiwan)—Not used • ZAF (South Africa)—District **Note:** If you enter addresses with the Andorran country code (AND), only Andorran addresses are returned, including the AND country code. If you enter an Andorran with a country code of

columnName	Description / Valid Values
	Spain (ESP), you will still get Andorran candidates, but with the ESP country code.
FirmName	Company or place name. For example: **Pitney Bowes** 4360 Dukes Rd Kalgoorlie WA 6430 **Aravali Vihar Police Station** Alwar 301030
HouseNumber	The building number. You may get better parsing results for some countries if you put the house number in this field instead of AddressLine1. For the best results, use the HouseNumber input field for these countries: India, Singapore, Malaysia, Thailand, and Turkey when using multi-line input. **Note:** The house number specified in the HouseNumber field takes precedence over any house number specified in the AddressLine1 field.
LastLine	The last line of the address. 4360 DUKES RD **KALGOORLIE WA 6430** 9, rue Paul Lafargue **93217 ST DENIS CEDEX**
Locality	The meaning of locality varies by country: • ARG (Argentina)—Neighborhood or barrio • AUS (Australia)—Not used • AUT (Austria)—Not used • BEL (Belgium)—Not used • BRA (Brazil)—Locality • CAN (Canada)—Dissemination Area and Enumeration Area (DA and EA) • CHE (Switzerland)—Not used • CHL (Chile)—Locality • CHN (China)—Locality • COL (Columbia)—Locality TBD • CZE (Czech Republic)—Locality

columnName	Description / Valid Values
	• DEU (Germany)—Not used • DNK (Denmark)—Not used • ESP (Spain)—Locality • EST (Estonia)—Locality • FIN (Finland)—Not used • FRA (France)—Not used • GBR (Great Britain)—Locality • GRC (Greece) —Locality • HKG (Hong Kong) —Locality • HUN (Hungary)—Locality • IDN (Indonesia)—Village • IND (India)—Locality • IRL (Ireland)—Not used • ITA (Italy)—Locality • JPN (Japan)—City district (chome) • LTU (Lithuania)—Locality • LVA (Latvia)—Locality • MAC (Macau) —Locality • MEX (Mexico)—Locality • MYS (Malaysia)—Not used • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Suburb • PHL (Philippines)—Locality • POL (Poland)—Not used • PRT (Portugal)—Locality • RUS (Russia)—Locality • SGP (Singapore)—Not used • SVK (Slovakia)—Locality • SVN (Slovenia)—Locality • SWE (Sweden)—Not used • THA (Thailand)—Not used • TUR (Turkey)—Locality • TWN (Taiwan)—Locality • ZAF (South Africa)—Locality Geocode Address Global can geocode addresses in which the Locality is input without the City. For example, the following addresses are geocoded identically: calle nicaragua 4705, palermo, capital federal calle nicaragua 4705, palermo

columnName	Description / Valid Values
PostalCode	The postal code in the appropriate format for the country. Australia uses a four-digit postal code system. In general, the first digit represents a state or territory, the second digit represents a region with a state, and digits three and four representing towns. For more information, see **Address Guidelines for Australia** on page 361. Austria uses a four-digit postal code system. The first two numbers indicate the sector and the last two numbers designate the delivery point within the sector. Belgium and Luxembourg use a four-digit postal code. The first two digits designate the sorting area (with the first digit usually representing the region) The next two digits represent the post office and delivery office. For Brazil, use the complete eight-digit postcode for the most accurate results (a single close match candidate). However, you can use a five-digit postcode. Switzerland uses a four-digit postal code beginning with a number between 1 and 9. Liechtenstein postcodes range from 9480 to 9499. Chile uses a seven-digit postal code beginning with a number between 1 and 9. Canada uses a six-character postal code. The first three characters are typically separated from the second three with a space. The first three characters are the FSA, the second three are the LDU. Street address geocoding only requires the FSA while postal code geocoding requires the full postal code (FSALDU). Choose whether you wish to have a space between the first three and last three characters of the postal code. Keeping this consistent speeds up the geocoding process. China has a six-digit postcode system. The first two digits indicate the province. The third digit and fourth digits indicate the district and city/town. The final two digits represent the postal delivery zone or prominent location. Larger provinces or cities might be assigned more than one block of codes. For example, Guangdong Province is assigned 51 and 52 as the first two digits. **Note:** For China, postal centroid geocoding requires the complete six-digit postcode. However, when a postal code is provided as part of an address for street geocoding, only four-digit postal codes are accepted and returned. Columbia uses a TBD postal code system

columnName	Description / Valid Values
	Czech Republic has a five-digit postal code system. Postal codes are typically separated by a space between the third and fourth numbers, but variations in spacing or no spacing in postal codes is supported. Denmark uses a four-digit postal code beginning with a number between 1 and 9. Spanish postcodes are composed of five digits (Andorra postcodes. are in the format AN###.) You must enter a complete postcode. Estonia uses a five-digit postal code system. Finland uses a five-digit postal code. The first two digits designate the post town or municipal area. The last three digits represent the destination post office. France uses a five-digit postal code. You must enter a complete postcode. The first two digits usually represent the department. The digits 00 represent military addresses and there are also special digit for overseas territories. The last three digits represent the local delivery area. In the larger cities (Paris, Lyon Marseille), the last two digits represent the arrondissement. For example, in the postcode: 33380, 33 is the department 380 is the delivery area. Greece uses a five-digit postal code system. Hungary has a four-digit postal code system. The first digit is for the postal region. Indonesia has a five-digit postal code system. India has a six-digit postal code system. In Ireland, postal data is available only for Dublin. For street-level geocoding, enter the full postcode in the format Dublin#, where # ranges from 1-24 (and one district known as Dublin6W). For postal centroid geocoding (or postal centroid fallback) use only the postal number as input. For example, specify the PostalCode as 3 rather than Dublin3. Only the postcode number is returned for postal centroid geocoding or fallback. Italy uses a five-digit postal code system. The first three numbers indicate the province and the last two numbers designate the delivery point. Japan uses a seven-digit numeric postal code system in the format: 999-99999. Latvia uses a four-digit postal code system. The first two numbers indicate the routing area and the last two numbers designate the post office. Postal codes cannot begin with 0, 22-29, 35, 49, 55 or 58-99.

columnName	Description / Valid Values
	Lithuania uses a five-digit postal code system. The LT prefix is allowed but is not required. Malaysia uses a five-digit postal code system. Netherlands uses a postal code system of four digits followed by two letters. The first two numbers indicate the city and region. The last two numbers and two letters designate the specific house number range. The letters represent a distribution code. You can geocode using the four digits only or you can include the two-letter distribution code. Norway uses a four-digit postal code. The first two digits designate the geographic area. New Zealand uses a four-digit postal code. The first digit represents the geographic region. The second and third digits represent the postal sort area. The last digit represents a specific urban area, rural delivery or PO box lobby. Poland uses a five-digit postal code in the format 99-999. The first digit represents the postal district. The second digit represents a major subdivision of that district. The three numbers after the dash represent the delivery post office. For large municipalities, the last three digits can represent a particular street, section of a street, or sometimes a specific address. Portugal uses a four-digit postal code beginning with a number between 1 and 9. More recently, Portugal instituted a seven-digit postcode with a dash and three additional digits following the first four digits. The geocoder ignores the additional three digits and returns the four-digit postal code. Singapore uses a six-digit postal code system. The first two numbers indicate the sector and the last four numbers designate the delivery point within the sector. Every building in Singapore has a unique postal code. Slovakia has a five-digit postal code system. Slovenia uses a four-digit postal code system. Sweden uses a five-digit postal code beginning with a number between 1 and 9. There is typically a space between the first three digits (the outward sorting part of the postcode) and the last two digits (the inward sorting part). Thailand uses a five-digit postal code system. Turkey uses a five-digit postal code system. Postal codes are typically separated by a space between the third and fourth numbers, but the geocoder can handle variations in spacing or no spacing in postal codes.

columnName	Description / Valid Values
	Taiwan uses a three-digit postal code system. In the United Kingdom, each address is associated with an alphanumeric code up to seven characters in length. Each postcode includes an average of 15 addresses. In some cases, where a customer receives a substantial amount of mail, for example, a business, the postcode pertains to just that one address (a large-user postcode).
StateProvince	The meaning of StateProvince varies by country: • ARG (Argentina)—Region or province • AUS (Australia)—State • AUT (Austria)—Region • BEL (Belgium)—Not used • BRA (Brazil)State— • CAN (Canada)—Province • CHE (Switzerland)—Not used • CHL (Chile)—State • CHN (China)—Province • COL (Columbia)—State TBD • CZE (Czech Republic)—Region name or alias. For example, the region HLAVNÍ MESTO PRAHA can be aliased as Prag. • DEU (Germany)—Bundesland • DNK (Denmark)—Not used • ESP (Spain)—Region • EST (Estonia)—Not used • FIN (Finland)—Region (län) • FRA (France)—Region • GBR (Great Britain)—Region • GRC (Greece) —Region • HKG (Hong Kong) —Province • HUN (Hungary)—County • IDN (Indonesia)—Province • IND (India)—State • IRL (Ireland)—Not used • ITA (Italy)—Region • JPN (Japan)—Prefecture • LTU (Lithuania)—Not used • LVA (Latvia)—State • MAC (Macau) — State • MEX (Mexico)—Province • MYS (Malaysia)—State (negeri)

columnName	Description / Valid Values
	• NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Region • PHL (Philippines)—State • POL (Poland)—Province (voivodship) • PRT (Portugal)—Region • RUS (Russia) —Region • SGP (Singapore)—Not used • SVK (Slovakia)—Not used • SVN (Slovenia)—Not used • SWE (Sweden)—Region (lan) • THA (Thailand)—Province (changwat) • TUR (Turkey)—Province • TWN (Taiwan)—County or Special Municipality • ZAF (South Africa)—State

Address Input Guidelines

Follow these suggestions to ensure that your street input data is in the best format possible for optimum geocoding.

Address Guidelines for Argentina

Follow these suggestions to ensure that your street input data is in the best format possible for optimum geocoding.

- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported in input addresses. Examples of typical thoroughfare types are:

Avendia	Avendia	Av	Ave	Avda
Calle	C	Clle		
Lateral Ruta	Lat Ruta	L R	LR	
Ruta Provincia	R P	RP		

- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. For example, you can input Calle 5 or Calle cinco and get the same returned candidates. Ordinals are also recognized in input addresses. The following numbers and equivalents are recognized as part of a street name input:

  ```
  1,UNO,PRIMERO,PRIMER,PRIMERA
  ```

  ```
  5,CINCO,QUINTO,QUINTA
  ```

For example, an input street name of "25 de Mayo" is recognized and handled the same way as "Veinticinco de Mayo".

- **Proper names and dates in street and town names**—Proper names and dates are often used in Argentina addresses, and these are handled by MapMarker Argentina. For example, an input street name of "Juan F. Ibarra" is recognized and handled the same way as "Juan Felipe Ibarra".
- **Directionals in addresses**—The following directionals are recognized in input addresses: Norte, Oriente, Este, Sur, Oueste, Occidente, Poniente, N, E, S, O, NE, NO, SE, SO, Noreste, Sudeste, Noroeste, Sudoeste.
- **Abbreviations in addresses**—A number of common abbreviations can be used in input addresses. GeocodeAddressARG will recognize the abbreviations and geocode successfully. For example, following is a small sample of equivalent abbreviations. This is not a complete list of address abbreviations.

Bario	BAR	
Ciudad	CD	CD.
Colonia	COL	COL.
Doctor	DR	
Francisco	FCO	
Manza	MZA	

Address Guidelines for Australia

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Australia addresses, see the Australia Post website: **www.auspost.com.au**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **House numbers and apartments**—Remove spaces between house number and apartment letter. 123 A Main Street does not geocode correctly because the geocoder assumes that the name of the street is A. Two workaround options are available:
  - Do not include the apartment letter.
  - Delete the space between house number and apartment letter: 123A Main Street geocodes because the geocoder ignores the A.
- **House numbers and unit information**—The house number pinpoints the location of the address. Unit input can be in one of two formats, as shown in the following examples:
  - Flat 2, 17 Jones St.
  - Apt 19, 123 Main St., where Apt is the unit type and 99 is the unit number. In this format, you must specify a valid unit type, otherwise the address will not be geocoded correctly. For a listing of valid unit types, see **www.auspost.com.au**.
  - 99-123 Main St. For an address derived from the G-NAF database, this address is a unique house number and is geocoded as a single delivery point, not as a range.
- **Directional suffixes**—Use directional suffixes wherever possible. This is especially true in towns and cities that consist mainly of numbered streets. These streets can only be distinguished by their directional

suffixes and street types. They also distinguish addresses on streets that change direction. For example: 123 Queen St W and 123 Queen St E would have very different coordinates.

- **Street types**—These distinguish different streets of the same name. For example, Main Avenue and Main Street are two entirely different entities. Using types is not essential, but it adds precision to your data. For a listing of street types, see **www.auspost.com.au**.
- **City/suburb name**—Enter the city/suburb name in the City field. Note that all input addresses must contain either a city or a postal code. If the geocoder does not make a close match on the street in the specified suburb, it can make a close match on the Local Government Area (LGA). Local Government Area (LGAs) do not encompass all of Australia. For example, LGAs do not cover extensive northern parts of South Australia, a large part of the Northern Territory, and the Australian Capital Territory. An LGA can include a number of official suburbs. It is best to use the suburb name for geocoding purposes, but it is possible to get a match on the LGA (or to return LGA information) in geocoded results.
- **Postal code**—All postcodes consist of four digits. While there are exceptions, the general format of postcodes is as follows:

  Note: These are general guidelines, and there are exceptions to these postcode number ranges.

  - Digit 1 represents the state or territory, within the following general conventions:

2	NSW
2600 and 2900	ACT
3	VIC
4	QLD
5	SA
6	WA
7	TAS
0	NT

  - Digit 2 represents a region within the state. State or territorial capitals usually have a 0 or 1 as the second digit.
  - Digits 3 and 4 represent towns. Major towns usually have a 0 as the last digit (or sometimes as the last two digits).

Address Guidelines for Austria

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Austrian addresses, see the Austria Post website: **www.post.at**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **State abbreviations**—State abbreviations are supported. The following table lists the Austrian states and their abbreviations.

Burgenland	Bgl
Kärnten	Ktn
Niederösterreich	NÖ
Oberösterreich	OÖ
Salzburg	Sbg
Steiermark	Stm
Tirol	Tirol
Vorarlberg	Vbg
Wien	Wien

- **Thoroughfare types**—Austrian thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.
- **Common words and abbreviations**—Common words, directionals, house number indicators, and abbreviations used in addresses are recognized.

Address Guidelines for Belgium

The Belgium geocoder supports locations in Belgium and Luxembourg. Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Belgium and Luxembourg addresses, see the Belgium Post website: **www.bpost.be**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Supported languages**—Dutch, French, and German language aliases and address formats are supported.
- **Thoroughfare types**—Belgian thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Dutch, French, and German thoroughfare types are also supported.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.
- **Common words and abbreviations**—Common words, directionals, house number indicators, and abbreviations used in addresses are supported.

Address Guidelines for Brazil

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types (pre and post thoroughfare types) and their common abbreviations are recognized and fully supported on input and output. Following is a partial list of recognized thoroughfare types.

  ALAMEDA=AL,ALAMEDA,ALUA LALA
  ACESSO=AC,ACESSO
  ARCO=ARCO
  AUTO-ESTRADA=AUTO-EST,AUTO-ESTRADA
  AVENIDA=AV,AVDA,AVE,AVENIDA
  AZINHAGA=AZINHAGA
  BAIRRO=BAI,BAIRRO
  BALUARTE=BALUARTE
  BECO=BECO
  Many others are also recognized.

- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses. For example, the following are all recognized in an input address:

  um, first, one, 1st, primera, primeiras, primeiro
  dois, second, two, 2nd, segunda, segundos
  Many other numeric designations are also recognized.

- **Directionals in addresses**—The following directionals are recognized in input addresses: norte, do norte, setentrional, sul, do sul, meridional, leste, este, do leste, do este, oriental, oeste, do oeste, ocidental
- **Common words and abbreviations**—Common words, directionals, house number indicators, and abbreviations used in addresses are supported. This is a partial list of common words and abbreviations that are recognized. Many other common words are also handled.

Articles of speech	dos, das, do, da, de, d, dom, uma, e, o, n, sn, sa, ie a, os, as, ao, em, no, na, nos, nas, ao, andar, andares, ander, apart, apartment, flat, apartmento, apto, apt, ap, ato, casa, cs, cob, unit (and many other common articles of speech)
Common words	os, nova, novas, na, no, nas, nos, um
Common abbreviations	aeroporto=aerop internacio=intern,int internacional=int international=int conselheiro=cnso desembargador=des regente=reg

limitado=ltda,ltdo,ltd
(and many other abbreviations)

Address Guidelines for Canada

Follow these suggestions to ensure that your street input data is in the best format possible for optimum geocoding.

- **Post Office Box numbers**—P.O. Boxes and Rural Route addresses are not geocodable. GeocodeAddressCAN will honor this user input and attempt to output Boxes and Rural Route information, but it will not be used for geocoding purposes.
- **Highway addresses**—Highway addresses (such as Hwy 401) are geocoded.
- **House numbers and apartment letters**—Remove spaces between house number and apartment letter. 123 A Main Street does not geocode correctly because the geocoder assumes that the name of the street is A. Two workaround options are available:
- Do not include the apartment letter.
- Delete the space between house number and apartment letter: 123A Main Street geocodes because the geocoder ignores the A.
- **House numbers and unit information**—The house number pinpoints the location of the address. Unit input can be in one of two formats, as shown in the following examples:
- 99-123 Main St, where 99 is the unit number. In this format, do not specify the unit type. The number 99 could be a suite, apartment, unit, floor or any valid unit type.
- 123 Main St. Apt 99, where Apt is the unit type and 99 is the unit number. In this format, you must specify a valid unit type (such as Apt or Suite), otherwise the address will not be geocoded correctly.
- **Directional suffixes**—Use directional suffixes wherever possible. This is especially true in towns and cities, such as Calgary, which consist mainly of numbered streets. These streets can only be distinguished by their directional suffixes and street types. They also distinguish addresses on streets that change direction. For example: 123 Main St W and 123 Main St E have very different coordinates.
- **Street types**—These distinguish different streets of the same name. For example, Main Avenue and Main Street are two entirely different entities. Using types is not essential, but it adds precision to your data. For a list of valid street types see **www.canadapost.com**.
- **City name**—Do not abbreviate city names. If the city is unknown, you may leave it blank, although this may affect the accuracy of the geocode.
- **Province name**—You may use the full province name (for example, Ontario), but using the two-character abbreviation (ON) is recommended to reduce the likelihood of input errors.

Address Guidelines for Czech Republic

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Aliases for town, district, and region names**—Aliases for town, district, and region names are supported.
- **Thoroughfare types**—Thoroughfare types (pre and post thoroughfare types) and their common abbreviations are recognized and fully supported on input and output.

Address Guidelines for Denmark

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Denmark postal system, see the Post Danmark website: **www.postdanmark.dk**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Supported languages**—Danish and English language aliases are supported for major town/municipality names. For example, Copenhagen (English) is equivalent to København (Danish).
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Following is a list of recognized thoroughfare types. This is not a complete list. Additional thoroughfare types may also be recognized.

  ALLÉ=alle
  ANLAEG=anlaeg
  ANLEAG=anleag
  BAKKEN=bakken
  BANEN=banen
  BASTION=bastion
  BOUL=boulevard
  BOULEVARD=bulevardi
  BRO=bro
  BROEN=bro
  BUEN=buen
  BULEVARDI=bulevardi
  DAMMEN=dammen
  DOSSERING=dossering
  GAARD=garrd
  GADE=gade
  GANGEN=gangen
  GARD=gard
  GÅRD=gard
  GET=get
  HAVE=have
  HAVN=havn
  HOEJEN=hojen
  HOJEN=højen
  HøJEN=højen
  HOLMEN=holmen
  HUSET=huset
  KAER=kaer
  KEAR=kear
  KRAENTEN=kraeten
  KREANTEN=kreanten
  LAENGEN=laengen
  LEANGEN=leangen
  MARKEN=marken
  PARK=parken

PARKEN=parken
PASSAGEN=passagen
PLADS=plads
SIDEN=siden
STIEN=stien
STRAEDE=straede
STREADE=streade
SVINGET=svinget
TOFTEN=toften
TORV=torv
VAENGE=vaenge
VANGEN=vangen
VARDEN=varden
VEANGE=veange
VEJ=vej

- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.
- **Common words and abbreviations**—Common words, directionals, house number indicators, and abbreviations used in addresses are supported.

Address Guidelines for Finland

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Finland postal system, see the Posti kuluttajille website: **www.posti.fi**. Finnish postal codes are managed by Itella Corporation. For information, see the **Itella postal codes page**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Finnish thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Swedish thoroughfare types are also supported.
- **Common words and abbreviations**—Common words, directionals, house number indicators, and abbreviations used in addresses are supported.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for France

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the French postal system, see the La Poste website: **www.laposte.com**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Virtual town names**—Some areas are generally recognized as cities even though they are not truly administrative cities. These areas represent Artificial City Areas, or Virtual Towns. Since these virtual town names are commonly used by the public, they are supported and treated as aliases for any of the encompassed towns. Returned candidates have the correct real town in place of the input virtual town.

Table 89: Virtual Towns in France

Virtual Town Alias	Encompassed Real Towns
Défense (La)	Part of: Nanterre, Puteaux, Courbevoie
Sophia Antipolis	Part of: Valbonne, Mougins, Vallauris, Antibes, Biot
Cergy-Pontoise	Menucourt, Courdimanche, Puiseux-Pontoise, Osny, Pontoise, Cergy, Vauréal, Neuville-sur-Oise, Saint-Ouenl'Aumône, Jouy-le-Moutier, Eragny
Marne-la-Vallée	Bry-sur-Marne, Villiers-sur-Marne, Noisy-le-Grand, Champs-sur-Marne, Emerainville, Noisiel, Lognes, Croissy-Beaubourg, Torcy, Collégien, Ferrières, Bussy- Saint-Georges, Bussy-Saint-Martin, Saint-Thibault-des-Vignes, Gouvernes, Conches, Guermantes, Jossigny, Lagny-sur-Marne, Montévrain, Chanteloup-en-Brie, Serris, Chessy, Coupvray, Magny-le-Hongre, Bailly- Romainvilliers
Saint-Quentin-en-Yvelines	Elancourt, Verrière (La), Trappes, Montigny-le-Bretonneux, Guyancourt, Voisins-le-Bretonneux, Magnyle- Hameau
Sénart	Tigery, Combs-la-Ville, Lieusaint, Moissy-Cramayel, Saint-Pierre-du-Perray, Savigny-le-Temple, Réau, Nandy, Cesson, Vert-Saint-Denis
Evry	Evry, Bondoufle, Courcouronnes, Lisses
Etang de Berre	Fos-sur-Mer, Miramas, Vitrolles, Istres
Isle-d'Abeau	Four, Isle d'Abeau (L'), Saint-Quentin-Fallavier, Vaulx-le- Milieu, Villefontaine

- **Common words and abbreviations**—The geocoder handles common abbreviations that are used in French addresses. It supports all the official French street type abbreviations plus a number of unofficial street types to help improve geocoding efficiency. A partial list is:

Table 90: Common French Address Abbreviations

Street Type or Name	Abbreviation
Hauts	No abbreviation.
appartement	APP, APT, APPART
Saint	ST
Sainte	STE
rue	r
Charles de Gaulle	CDG
Regiment D'Infanterie de Marine	RIMA
Division Blindée	DB

- **Directionals in addresses**—Abbreviated street directionals are also handled on input and the returned candidate displays the complete directional.

Table 91: Street Directionals

N	N.	Nord
S	S.	Sud
E	E.	Est
O	O.	Ouest
NE	N.E.	Nord-Est
SE	S.E.	Sud-Est
NO	N.O.	Nord-Ouest
SO	S.O.	Sud-Ouest

- **Ordinals and numbered street names**—Input addresses can include ordinals such as 1er, 2e, 2nd, 2nde, 3e. All subsequent ordinal street names are designated with "e" or "ème". You can also specify

numbers in street names or express the numbers as words. For example, the following street names are equivalent and can both be geocoded as part of an input address:

```
Rue du 4 septembre
Rue du quatre septembre
```

- **House numbers with letters**—House numbers can include letters, such as `85B Ave des provinces`.
- **Postal box (BP) addresses**—The geocoder can handle Postal Box (Boite Postale) addresses. For the following input address, a close match candidate is returned with a result code of S5HPNTSC-A.

  AddressLine1: `BP 112 2 Avenue CDG`
  PostalCode: `78150`
  City: `Le Chesnay`

  Note that in this example, the street name `CDG` is returned as `Charles de Gaulle` and the postcode is corrected. The BP itself is not returned.
- **CEDEX addresses**—The geocoder does not use CEDEX for geocoding but CEDEX will not interfere with geocoding. CEDEX can be entered in AddressLine1, City, or PostalCode fields. The CEDEX itself is not returned but the complete postcode is returned. For the following input address, a close match candidate is returned with a result code of S5HPNTS--A.

  AddressLine1: `17 Rue Louise Michel`
  PostalCode: `92301 CEDEX`
  City: `Levallois-Perret`

  The postcode is returned but not a postal centroid (there is no Z in the ninth position of the return code). The CEDEX itself is not returned.
- **Paris address formats and arrondissements**—Paris addresses typically have a different input format. The house number appears after the street name rather than before the street name. The geocoder handles this input format and geocodes correctly. Arrondissements (the last two digits of the postcode) can be entered and the complete locality and postcode information is returned. For the following input address, a close match candidate is returned with a result code of S5HPNTSCZA.

  AddressLine1: `7 Rue Beranger`
  PostalCode: `75003`
  City: `Paris`

  The returned Locality field includes the arrondissement (district) information. The Paris region includes 20 arrondissements, which are represented by the last two digit of the postcode (the first three digits are 750). A Paris address may be written with the last two digits only. For the following input address, a close match candidate is returned with a result code of S5HPNTSC-A

  AddressLine1: `51 Rue Lafitte`
  PostalCode: `09`
  City: `Paris`

  The complete postcode (75009) is returned even though 09 (representing the ninth arrondissement) was entered.
- **Military addresses**—Military addresses (including typical military address abbreviations) are handled. The first two digits usually represent the department. The digits 00 represent military addresses.

- **Monaco addresses**—The geocoder handles Monaco addresses. Specify Monaco in the StateProvince input field. All Monaco postcodes begin with the number 98. Monaco is returned in the address even if it was not specified on input.

Address Guidelines for Germany

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the German postal system, see the Deutsche Post website: **www.deutschepost.de**.

Addresses must contain either a city or a postal code.

German addresses may contain some or all of the following address elements.

Table 92: German Address Elements

Address Element	Field	Example
Street address	AddressLine1	Muldenweg 2
Ort	City	Vaihingen an der Enz
Kreis	County	Ludwigsburg
Bundesland	StateProvince	Baden-Württemberg
Postcode	PostalCode	71665

Address Guidelines for Greece

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Greece addresses, see the Hellenic Post website: **http://www.elta.gr**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.

Address Guidelines for Hong Kong

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Hong Kong addresses, see guidelines on the Hong Kong postal website **http://www.hongkongpost.com/eng/publications/guide/content/6.1.pdf**.

- **Required fields** — Addresses must contain a city. HKG does not consider postal codes in addresses.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported in input addresses.

- **Common words and abbreviations** — Common words, directionals, house number indicators, and abbreviations used in addresses are supported.
- **Numbers and numeric equivalents**—Numbered streets are mapped to the named equivalents.

Address Guidelines for Hungary

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Hungarian postal system, see the Magyar Posta website: **www.posta.hu**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **House number, floor number, and apartment numbers**—The geocoder supports street addresses with house number, floor, and apartment information. For example:

  AddressLine1: `Kavics utca 13 II/3`
  PostalCode: `1025`
  City: `Budapest`

  This is interpreted as house number 13, floor Roman numeral 2, apartment number 13. This could also be input as: `Kavics utca 13 II em 3 ajtó`.

Address Guidelines for Ireland

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Ireland postal system, see the An Post website: **www.anpost.ie**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types (pre and post thoroughfare types) and their common abbreviations are recognized and fully supported on input and output. Following is a partial list of recognized thoroughfare types. Many others are also recognized.

Table 93: Thoroughfare Types in Ireland

Pre Thoroughfare Types	Post Thoroughfare Types
Áirse	Street (Also: St.,ST,STR)
Ard	Terrace (Also: TCE)
Arda	Third
Ardán	Track (Also: TRK,TCK)
Ascaill	Vale
Barra	Valley

Pre Thoroughfare Types	Post Thoroughfare Types
Bealach	View
Bogha	Village
Bóithrín	Walk (Also: WK)
Bóthar	Way (Also: WY)
Brí	Wood(s)
Bruach	Yard (Also: Yd.)
Búlbhard	

- **Common words and directionals**—The geocoder recognizes common words and directionals used in addresses and can geocode these addresses successfully. This is a partial list of common words and directionals that are understood by the geocoder. Many other common words are also handled.

  an, na, a
  Cuar, Circular
  Cros, Cross
  Thoir, East
  Tosaigh, Front
  Laistigh, Inner
  Beag, Little
  Lower
  Main
  Middle
  Military
  Nua, New
  Thuaidh, North
  Sean, Old
  Lasmuigh, Outer
  ear
  Dara, Second
  Theas, South
  Third
  Uachtarach, Uacht, Upper
  Thiar, West
  Iartharach, Western
  Saint, St, Naomh
  retail, park, business, center, centre, shopping, office, mills,
  shop, service, outlet
  saint, station

The
Park
House
Building, Blgs
House, Hse, Hses
branch

- **Directionals**—The geocoder recognizes the following directionals: North, N, Nth, South, S, Sth, East, E, Est, West, W, Wst, NE, NW Sea SW Lower, LW, LR, Upper, UP, Upp, Uppe, upr, Thuaidh, Thoirm, Thiar, Theas
- **Common abbreviations**—The geocoder recognizes common abbreviations used in addresses and can geocode these addresses successfully. This is a partial list of abbreviations that are understood by the geocoder. Many other common abbreviations are also handled.

Table 94: Common Abbreviations

Word	Abbreviation
saint	st., st
great	gt., gt
north	n, nth
south	s, sth
east	e, est
west	w, wst
northeast	ne
northwest	nw
southeast	se
southwest	sw
lower	lw, lr
upper	up, upp, uppe
mount	mnt, mt
and	&

Word	Abbreviation
football	f
club	c
limited	ltd
park	pk
estate	est
gardens	gdns
building	bld
industrial	ind
industries	ind
number	num
center	cnt, centre
centre	cnt, center
country	co
market	mrkt
square	s
o+connell	oconnell

- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses. For example, the following are all recognized in an input address: um, first, one, 1st, primera, primeiras, primeiro dois, second, two, 2nd, segunda, segundos.

Address Guidelines for Italy

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Italy postal system, see the Posteitaliane website: **www.poste.it**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **German language addresses**—German address formats (common in the South Tyrol area of Italy) are handled and geocoded correctly. Typical German thoroughfare types and abbreviations are supported. For example, the street name Marienstraße could be abbreviated as Marienstr, and the same candidate is returned. Note that regardless of whether strasse or straße is entered as input, strasse is returned in the output candidate.
- **Aliases for regions, localities, and provinces**—Aliases can be used on input. For example, Tuscany is an alias for the region of Toscana. When you geocode, the returned candidate matches the user input. That is, if aliases were used then aliases are returned.
- **Regions and provinces**—For street geocoding, region names (which are entered in the StateProvince field) are not used for geocoding purposes, but are returned. Province abbreviations consisting of two letters are returned in the County field. Italy has 20 regions and 110 provinces.
- **PO boxes**—Post Office Box numbers are not used for address matching or geocoding purposes, but this does not interfere with matching or geocoding. The PO Box information is not returned. The following formats are recognized:

  Casella Postale
  CP

- **Thoroughfare types**—Thoroughfare types (pre and post thoroughfare types) and their common abbreviations are recognized and fully supported on input and output. Both Italian and German thoroughfare formats are supported.
- **Common words, abbreviations, and directionals**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. For example, if you enter the street name `Via 42 Martiri`, the street name `QUARANTADUE MARTIRI` is returned. Ordinals are also recognized in input addresses.

Address Guidelines for Japan

A typical Japanese address looks like this:

北海道札幌市中央区大通西２８丁目３番２２号

The elements of this address are described in the following table.

Table 95: Japanese Address Elements

Address Element	Spectrum Field Name	Example
Prefecture	StateProvince	北海道
City (Shi)	County	札幌市中央区

Address Element	Spectrum Field Name	Example
Municipality Subdivision (Oaza)	City	大通西
City District (Chome)	Locality	２８丁目
Block/lot number	AddressLine1	３番２２ Block and lot numbers are the most specific address elements. Japanese addresses do not typically have street names.

Address Guidelines for Macau

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Macau addresses, see the Macau Post website: **http://www.macaupost.gov.mo/**.

- **Required fields**—Addresses must contain a city. MAC does not consider postal codes in addresses..
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.

Address Guidelines for Malaysia

For information on Malaysian addresses, see the Pos Malaysia website: **www.pos.com.my**.

Some Malaysian addresses do not require a city or a postal code. For certain addresses, the geocoder can obtain a geocode by using only street information, which can be a combination of address number and street name, without any town or postal code.

Address Guidelines for Mexico

Follow these suggestions to ensure that your street input data is in the best format possible for optimum geocoding.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Aliases for cities**—You can input aliases for city names and still get a close match. For instance, you could enter "Coyoacán" or "Miguel Hidalgo" and it would match to Mexico city.

Table 96: Aliases for Mexican Cities

Alias	Town
Mexico City/Ciudad de Mexico	La Magdalena Contreras Álvaro Obregón Azcapotzalco Benito Juárez Ciudad Madero Coyoacán Cuajimalpa de Morelos Cuauhtémoc Gustavo A. Madero Iztacalco Iztapalapa Miguel Hidalgo Milpa Alta Tláhuac Tlalpan Venustiano Carranza Xochimilco
Ciudad López Mateos	Atizapán de Zaragoza
Las Tortugas	Emiliano Zapata
Naucalpan	Naucalpan de Juárez
San Bernardo	San Bernardo Mixtepec

- **Aliases for states**—You can use state aliases and get a close match. For example, if you enter "YUC" it would match to Yucatán.

Table 97: Aliases for Mexican States

State	Alias
Aguascalientes	AGS/AG/AGU
Baja California	Bassa California/Neder-Californië/BC/BJ/BN/BAJ/B C
Baja California Sur	Bassa California del Sud/BCS/BS/BAS/B C S

State	Alias
Campeche	CAM/CP/CM
Chiapas	CHIS/CH/CU/CHP
Chihuahua	CHIH/CI/CL/CHU
Colima	COL/CL/CH
Coahuila de Zaragoza	COAH/CU/CS/COA/CZ/C Z/Coahuila
Distrito Federal	Distretto Federale/DF/MDF/D F
Durango	DGO/DG/DUR
Guanajuato	GTO/GJ/GT/GUA
Guerrero	GRO/GR/GUE
Hidalgo	HGO/HG/HID
Jalisco	JAL/JA
México	Mexico/Mexiko/Meksiko/Messico/MEX/EM/MX
Michoacán de Ocampo	MICH/MH/MC/MIC/MO/M O/Michoacan
Morelos	MOR/MR
Nayarit	NAY/NA
Nuevo León	NL/NUE/N L
Oaxaca	OAX/OA
Puebla	PUE/PU/PUB
Querétaro Arteaga	QRO/QA/QE/QDA/Q A/Queretaro
Quintana Roo	QROO/QR/QI/QRO/Q R/Q Roo
San Luis Potosí	San Luis Potosí

State	Alias
Sinaloa	SIN/SI
Sonora	SON/SO
Tabasco	TAB/TA/TB
Tamaulipas	TAMPS/TM/TAM
Tlaxcala	TLAX/TL/TLX
Veracruz de Ignacio de la Llave	VER/VZ/VE/VCL/Veracruz
Yucatán	Yucatan/YUC/YC/YU
Zacatecas	ZAC/ZT/ZA

- **Numbers, numeric equivilants, and ordinals**—Numbered streets are mapped to the named equivalents. For example, you can input Calle 5 or Calle cinco and get the same returned candidates. Ordinals are also recognized in input addresses. For example, the following are all recognized in an input address: 5, CINCO, QUINTO, and QUINTA.
- **Directionals**—The following directionals are recognized in input addresses: Norte, Oriente, Este, Sur, Oueste, Occidente, Poniente, N, E, S, O, NE, NO, SE, SO, Noreste, Sudeste, Noroeste, and Sudoeste.

Address Guidelines for Netherlands

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Netherlands addresses, see the TNT Post website: **www.tntpost.nl**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Aliases for cities**—Locality, town, and province aliases can be used on input. When you geocode, the better matched input name (the official name or alias) is returned with the candidate.
- **Post office box numbers**—Post Office Box numbers are not used for address matching or geocoding purposes, but this does not interfere with matching or geocoding. The PO Box information is not returned. The following formats are recognized: Postbus, PostFach.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for New Zealand

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on New Zealand addresses, see the New Zealand Post website: **www.nzpost.co.nz**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Aliases for suburbs**—The geocoder supports locally used suburb names in addition to the officially recognized suburb names. For example, Rosedale is an alias of the official suburb name of Hargest.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for Norway

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Norway addresses, see the Posten Norge website: **www.posten.no**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Common words, abbreviations, and directionals**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully. Following is a list of common words that are recognized. Other other common words and abbreviations may also be recognized.

Common words	av, bak, bakerst de, dei, den, det e, ei, ein, eit, en, et for, foran, fra, gammel i, imellom, Industriebygget, inntil, kort lang, lille, lita, lite, liten. med, mellom nær, nærmest, nest, ny og, over, overfor, på små, stor, til, under ved, ved siden av
Common abbreviations	Sankt=St
Directionals	nord, øst, syd, vest

- **Numbers, equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses

Address Guidelines for Philippines

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Philippines addresses, see the Philippine postal website **http://www.philpost.gov.ph/**www.philpost.gov.ph/.

- **Required fields** — Addresses must contain a city. PHL does not consider postal codes in addresses.

- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported in input addresses.
- **Common words and abbreviations** — Common words, directionals, house number indicators, and abbreviations used in addresses are supported.
- **Numbers and numeric equivalents**—Numbered streets are mapped to the named equivalents.

Address Guidelines for Poland

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Poland addresses, see the Polish Post website: **www.poczta-polska.pl**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for Portugal

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Portugal addresses, see the CTT Portugal Post website: **www.ctt.pt**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for Russia

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Russian addresses, see the Russian Post website: **http://www.russianpost.ru/**

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.

Address Guidelines for Singapore

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Singapore addresses, see the Singapore Post website: **www.singpost.com**.

- **Required fields**—Addresses must contain either a city or a postal code.

- **PO box addresses**—Post Office Box numbers are not used for address matching or geocoding purposes, but this does not interfere with matching or geocoding. The PO Box information is not returned. The following formats are recognized: P O Box, Locked Bag Service.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output. The following table shows is a partial list of recognized thoroughfare types. Others may also be recognized.

Table 98: Thoroughfare Types

Pre-thoroughfare types	lorong=lorong, lrg, lor, lorang
	jalan=jalan, jln, jl
	lengkong=lengkong, lkg
	kallang=kallang
	mount=mount, mt
	upper=upper, upp
Post-thoroughare types	track=trk,tck
	street=st
	road=rd
	drive=dr
	crescent=cr,cres,crescent,cresent
	boulevard=bvd,blvd,bouleyard,boulvard
	hill=hill
	gate=gate
	mall=mall
	avenue=ave,av,avnue
	link=lk
	lane=l
	walk=wk
	green=grn
	highway=hwy
	quay=quay, qy
	parkway=pwy

- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.

This is a partial list of common words, abbreviations, and directionals that are recognized. Other common words may also be recognized.

Table 99: Common Words

Category	Words
Common words	and, ltd, international, industrial, plc, group, co, front, by, bay
Articles of speech	the, at, a, an, blk
Building identifiers	center, centre, building, house, place, court, galleria, hotel, park, complex, mart, temple, bank, exchange, station, community, area, mosque, depertment, department, post office, shrine, chambers, masjid, apartments, complex, bureau, resort, lodge, harbour monastery, convent, restaurant, golf course, estate, campus, institute, university, facility, tunnel, libraray, society, mansion, hub, beach, church, park, kiosk, mission, condominium, warehouse, hall, laboratories, hospital, corporation, fire post, terminal, workshop, headquarters, cemetery plaza, villa, garden, gardens, tower, station, hall, lodge, cottages, cottage, village, gurdwara, place
Common abbreviations	AYE=Ayer Rajah Expressway BKE=Bukit Timah Expressway CTE=Central Expressway ECP=East Coast Parkway KJE=Kranji Expressway KPE=Kallang-Paya Lebar Expressway PIE=Pan Island Expressway SLE=Seletar Expressway TPE=Tampines Expressway Ctrl=Central E=East S=South W=West

Category	Words
	N=North JLN=Jalan CR=Crescent GR=Grove L=Lane WK=Walk LRG=Lorong TG.=TANJONG
Directionals in addresses	North, N, Nth, South, S, Sth, East, E, Est, West, W, Wst, NE, NW, SE, SW, T1, T2

Address Guidelines for Spain

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information about the Denmark postal system, see the Spanish postal service website: **www.correos.es**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Supported languages**—If a street has a Spanish name and Basque or Catalan alternate name, the returned candidate street name will match the input. That is, if a Basque or Catalan street name is used on input, then the Basque or Catalan alternate street name is returned as a close match candidate. If the Spanish street name is input, the Spanish street name is returned.
- **Abbreviations in addresses**—The geocoder handles common abbreviations that are used in Spanish addresses. This includes abbreviations for building types, floor indicators, titles, and articles of speech. The geocoder also supports all the official Spanish street type abbreviations plus a number of unofficial street types to help improve geocoding efficiency. Abbreviated street directionals are also handled on input and the returned candidate displays the complete directional. For example, input of Arroya Guadalpia N returns the street Arroya Guadalpia Norte.

N,Norte, Nort	North
S,Sur, Sud	South
E,Este, Est	East
O,Oeste, Oest	West

Address Guidelines for Sweden

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Sweden addresses, see the Post Norden website: **www.posten.se**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Following is a list of recognized thoroughfare types. Other types may also be recognized.

  ALLE, ALLÉ, BACKE, BULEVARDI, BULEVARD
  DALLEN, ESPLANADI, GATAN, GATTAN, GATA
  GATE, GATEN, GRANDEN, GRAND, GRÄND
  GÅRD, GARD, KATU, KUJA, LERIG
  LIDEN, PARKEN, PARK, PLAN, PORTTI
  PROMENADEN, PROMENADER, RACE, RÅDE, RADE
  RINNE, STIGEN, STIG, STRAND, SKOG
  TIE, TORG, VAG, VAGEN, VÄGEN
  VÄG, VAGAN, VÄGAN

- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully. Following is a list of common words that are recognized. Other other common words and abbreviations may also be recognized.

  andra, at, åt, av, bak, bakom, de, del
  den, dens, emellan, en, for, för, fore, för
  framfor, framföre, framme, fran, från, gamla, gammal, gammalt
  i, in, infor, införe, intill, kort, lang, lång
  liten, litens, med, mellan, motliggande, narheten, närheten
  narmast, närmast, nord, Norra, ny, nya, och, ost
  Ostra, Östra, over, över, pa, på, sidan, sma
  små, Sodra, Södra, stor, syd, till, till, under
  vast, väst, Vastra, Västra, via

- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for Switzerland

The Switzerland geocoder supports locations in Switzerland and Liechtenstein. Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Swiss addresses, see the Swiss Post website: **www.swisspost.ch**. For additional information on the Liechtenstein postal system, see the Liechtenstein Post Corp website: **www.post.li**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—German, French, and Italian thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Over 300 thoroughfare types are recognized.
- **Common words and abbreviations**—You can use German, French, and Italian common words, directionals, house number indicators, and abbreviations that are typically used in addresses.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents in German, French, or Italian. Ordinals are also recognized in input addresses.

Address Guidelines for Thailand

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Thailand addresses, see the Thailand Post website: **www.thailandpost.com**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully.
- **Numbers, numeric equivalents, and ordinals**—Numbered streets are mapped to the named equivalents. Ordinals are also recognized in input addresses.

Address Guidelines for Turkey

Follow these guidelines to provide input that GeocodeAddressGlobal can successfully geocode. For additional information on Turkey addresses, see the Turkey PTT website: **www.ptt.gov.tr**.

- **Required fields**—Addresses must contain either a city or a postal code.
- **Thoroughfare types**—Thoroughfare types and their common abbreviations are recognized and fully supported on input and output. Examples of typical thoroughfare types and their abbreviations are: Bulvar, Bulvari (boulevard) Cadde, Caddesi, Cd, Cad (avenue, lane) Mahalle, Mahallesi, Mah (neighborhood, quarter) Sokak, Sk, Sokagi (street) Yolu, Yol (way, road) This is not a complete list. Other thoroughfare types are also recognized.
- **Common words and abbreviations**—The geocoder recognizes common words, directionals, house number indicators, and abbreviations used in addresses and can geocode these addresses successfully. This is a partial list of common words and abbreviations that are recognized. Many other common words are also handled.

Table 100: Common Words

Category	Words
Articles of speech	ve
Directionals	Words representing north, south, east, west, northwest, northeast, southeast, southwest
Buildings	bina, binasi, apartman ,kurma, yap; ev, konut, mesken ,ev halki, yurt;zemin, taban, pist, kat
Area names	semt, semti, ent, sehir, kasaba, sehir halki, büyük kasaba

Category	Words
Levels or positions	yukarida, asagida, asagi, altta, üzerinde, altinda, arkasinda, içinde, moda, ön, yanindaki, arasinda, sol, sag
House number indicators	hayir., rakam, sayi, numara, miktar
Common abbreviations	DR (Doctor), Prof (Professor), IST (Istanbul)

Single Line Input

Instead of entering each address element in separate fields, you may enter the entire address in the AddressLine1 input field.

For all countries except Japan, you can enter addresses in one or more of these single-line formats.

Note: Not all formats work for all countries.

```
StreetAddress;PostalCode;City
StreetAddress;City;PostalCode
StreetAddress;PostalCode
StreetAddress;City
StreetAddress,PostalCode,City,StateProvince
StreetAddress,PostalCode,City,County,StateProvince
StreetAddress;City;StateProvince;PostalCode
StreetAddress;City;StateProvince+
StreetAddress;City;County;PostalCode
StreetAddress;County;City
StreetAddress;County
StreetAddress,City,PostalCode,StateProvince
PostalCode,City,StateProvince,StreetAddress
PostalCode;City;StreetAddress
PostalCode;StreetAddress
PostalCode;StreetAddress;City
PostalCode;City
City;StreetAddress
City;PostalCode;StreetAddress
```

```
City,StateProvince,StreetAddress

City,PostalCode,StateProvince,StreetAddress

City;StreetAddress;Locality

StreetAddress;Locality

StreetAddress;Locality;City

StreetAddress;Locality;City;PostalCode
```

Note: U.K. addresses must be in this format. You must include either a city plus state/province or a postal code as the last address element.

```
StreetAddress;City;Locality

StreetAddress;City;StateProvince;Locality

StreetAddress;City;County;StateProvince;PostalCode

StreetAddress;PostalCode;City;StateProvince;Locality
```

Where:

- *StreetAddress* can be house number and street name in either order (with street type immediately before or after the street name).
- *City* is the town. For Indonesia and Thailand, this is the subdistrict.
- *Locality* is the locality name. For Indonesia, this is the village.
- *County* is the county name. For Chile, Philippines, and Thailand, this is the district. For Indonesia, this is the regency/city.
- *StateProvince* is the state, province, or equivalent administrative division.
- *PostalCode* is the complete postcode. For Brazil, *PostalCode* should be the complete eight-digit postcode for the most accurate results. However, you can use a five-digit postcode.

The matching accuracy for single line input is comparable to that of structured address input. The performance of single line input addresses may be slightly slower than that of structured address input.

For best results, use delimiters (comma, semicolon, or colon) between each component of the address. For example,

```
26 Wellington Street East;Toronto;ON;M5E 1S2

schulstrasse 22,Dresden,01328

Urión 30,Col. Tlatilco,02860,Mexico,D.F.
```

If the input address is missing delimiters, spaces are recognized as separators and internal parsing rules identify address components. In the example above, the address would still successfully geocode even if some or all of the delimiters were missing in the input.

Note: Non-delimited or partially-delimited single line addresses may take longer to geocode and may not produce the same results as delimited single line input. This is especially true for addresses with multi-word street names or cities. To optimize single line geocoding, use delimiters between address components (particularly between street name and city).

Punctuation is ignored for geocoding purposes.

Format for Japan

Japanese addresses are typically written in single line format, without any delimiters to separate address fields. The typical format is:

```
<prefecture><city><municipality subdivision><city district><block><lot><other>
```

where:

- prefecture = ken
- city = shi
- municipality subdivision = oaza
- city district = chome
- block = numbered city block (ban)
- lot = sub blocks or building number (go)
- other = building names, flat numbers, or other identifiers. This information is ignored by the Japan geocoder.

Note: Block and lot numbers are the most specific address elements in Japan. Japanese addresses typically do not have street names.

Table 101: Example Japanese Addresses

Address	Description
東京都渋谷区広尾1-1-39	Chome, block, and lot separated by a hyphens.
東京都渋谷区広尾1丁目1-39	Block and lot separated by hyphen, chome indicated by chome identifier.
東京都渋谷区広尾1丁目1番39号	Chome, block, and lot separated by their identifiers.

Guidelines for Single Line Input

- For Germany, France, and New Zealand, you must specify a value in either the PostalCode field or the City field in order for single line input to successfully geocode.
- For Canada, if you omit the postal code and country, the geocoder still geocodes the address based on street address, city, and province.
- The country is not required. Each country geocoder assumes that the address is in its country.
- Firm information (placename, building name, or government building) is returned if available.

Street Intersection Input

If you enter a street intersection as input, the geocoder will provide the coordinates of the intersection.

Note: Intersection geocoding is not available for the United Kingdom (GBR).

To enter an intersection, specify the two street names separated by a double ampersand (&&) in AddressLine1. For example:

AddressLine1: `Ocean Ave && New South Head Rd`
City: `Woollahra`

AddressLine1: `AVENIDA VINTE E OITO DE MARÇO && AVENIDA JOSÉ ALVES AZEVEDO`
PostalCode: `280`

AddressLine1: `Chemin du Prieuré && Rue de la Montcient`
City: `Seraincourt`

AddressLine1: `Akti Miaouli && Agias Marinas`
City: `Markopoulo Mesogaias`

Note: Do not use /, _, -, or any other separator. Intersections must be delimited using a double ampersand (&&) as a separator.

All close match criteria are enforced for intersection geocoding, just as for any street level geocoding.

Options

GeocodeAddressGlobal allows you to set default processing options through the Management Console. You can override certain settings for individual calls to GeocodeAddressGlobal using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

Geocoding Options

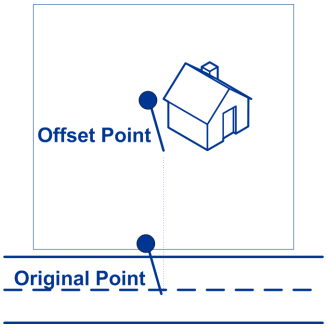
The following table lists the options that control how a location's coordinates are determined.

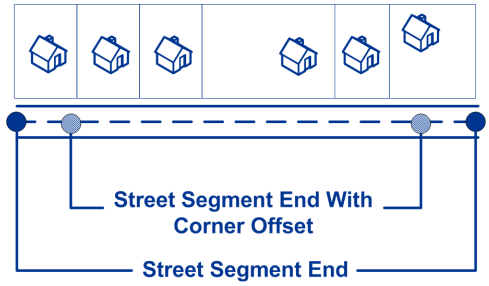
Table 102: Geocoding Options

optionName	Description / Valid Values	
GeocodeLevel	Specifies how precisely you want to geocode addresses. One of the following:	
	StreetAddress	The geocoder attempts to geocode addresses to a street address, but some matches may end up at a less precise location such as a postal code centroid, intersection, or shape path.
	PostalCenroid	The geocoder attempts to geocode addresses to the most precise postal code it finds. The advantage of postal code centroid matching is the speed of the operation. The disadvantage of postal code matching is that the geocoder only examines the PostalCode field. If you use street address precision, the geocoder looks at both the street name and the PostalCode field and attempts to return street-level coordinates and optionally fall back to postal code

optionName	Description / Valid Values
	coordinates. Postal centroid geocoding is not available for this country. **GeographicCentroid** The geocoder attempts to geocode addresses to the geographic centroid of a city or state. This option is not available for the United Kingdom (GBR).
Interpolation	Specifies whether to perform address point interpolation. This option only works if you are using both a point database and a street segment database for the specific country. This option is not available for all countries. This option is available for Australia, Canada, Germany, and Switzerland (including Liechtenstein). Address point interpolation uses point data to refine geocode results. By default, the geocoding process estimates the location of an address based on the street numbers at either end of street segment. For example, if a street segment runs from 100 Main St. to 200 Main St., then a request for 150 Main St. will return a location in the middle of the segment. With interpolation, the geocoder finds the position of 180 Main St. in the point data, and it is about two-thirds of the way down the street. Using this information, the geocoder can estimate the position of 150 Main St. based on 100 and 180 Main St. In this case, the geocoder estimates the location of the address slightly away from the center of the segment. One of the following: **Y** Yes, perform address point interpolation. **N** No, do not perform address point interpolation.
FallbackToGeographic	Specifies whether to attempt to determine a geographic region centroid when an address-level geocode cannot be determined. This option is not available for the United Kingdom (GBR). One of the following: **Y** Yes, determine a geographic centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a geographic centroid when an address-level centroid cannot be determined.
FallbackToPostal	Specifies whether to attempt to determine a postal code centroid when an address-level geocode cannot be determined. This option is not

optionName	Description / Valid Values
	available for Argentina, Hong Kong, Macau, Philippines, and South Africa. One of the following: **Y** Yes, determine a postal code centroid when an address-level centroid cannot be determined. Default. **N** No, do not determine a postal code centroid when an address-level centroid cannot be determined.
OffsetFromStreet	Indicates the offset distance from the street segments to use in street-level geocoding. The distance is specified in the units you specify in the OffsetUnits option. **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). The default value varies by country: • 10 meters—Australia (AUS), Austria (AUT), Germany (DEU). • 7 meters—Argentina (ARG), Belgium and Luxembourg (BEL), Brazil (BRA), Canada (CAN), Switzerland and Liechtenstein (CHE), Czech Republic (CZE), Chile (CHL), China (CHN), Denmark (DNK), Spain (ESP), Estonia (EST), Finland (FIN), France (FRA), United Kingdom (GBR), Greece (GRC), Hong Kong (HKG), Hungary (HUN), Indonesia (IDN), India (IND), Ireland (IRL), Italy (ITA), Latvia (LVA), Lithuania (LTU), Macau (MAC), Mexico (MEX), Malaysia (MYS), The Netherlands (NLD), Norway (NOR), New Zealand (NZL), Philippines (PHL) Poland (POL), Portugal (PRT), Russia (RUS), Singapore (SGP), Slovakia (SVK), Slovenia (SVN), Sweden (SWE), Thailand (THA), Turkey (TUR), Taiwan (TWN), South Africa (ZAF). The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point.

optionName	Description / Valid Values
	 Street coordinates are accurate to 1/10,000 of a degree and interpolated points are accurate to the millionths of a degree.
OffsetFromCorner	Specifies the distance to offset the street end points in street-level matching. The distance is specified in the units you specify in the OffsetUnits option.This value is used to prevent addresses at street corners from being given the same geocode as the intersection. **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). The default value varies by country: • 12 meters—Australia (AUS), Austria (AUT), Germany (DEU) • 7 meters—Argentina (ARG), Belgium and Luxembourg (BEL), Brazil (BRA), Canada (CAN), Switzerland and Liechtenstein (CHE), Chile (CHL), China (CHN), Czech Republic (CZE), Denmark (DNK), Spain (ESP), Estonia (EST), Finland (FIN), France (FRA), United Kingdom (GBR), Greece (GRC), Hong Kong (HKG), Hungary (HUN), Indonesia (IDN), India (IND), Ireland (IRL), Italy (ITA), Latvia (LVA), Lithuania (LTU), Macau (MAC) Mexico (MEX), Malaysia (MYS), The Netherlands (NLD), Norway (NOR), New Zealand (NZL), Poland (POL), Portugal (PRT), Russia (RUS), Singapore (SGP), Slovakia (SVK), Slovenia (SVN), Sweden (SWE), Thailand (THA), Turkey (TUR), Taiwan (TWN), South Africa (ZAF). The following diagram compares the end points of a street to offset end points.

optionName	Description / Valid Values
	
OffsetUnits	Specifies the unit of measurement for the street offset and corner offset options. One of the following: **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). • Feet • Miles • Meters • Kilometers The default is Meters.
GNAFPointType	This option is available for Australia only. For street address matching, specifies whether to return the parcel latitude/longitude or the street frontage latitude/longitude. This option is available only if you have the G-NAF database installed, and the database selected in the **Database** list on the Data tab includes the G-NAF database. This option only affects addresses matched to the G-NAF database. One of the following: **P** In a street address match, return the exact location of the parcel. This is the standard G-NAF point which is the exact authoritative point returned by the G-NAF database. Default. **S** In a street address match, return the street frontage point for the parcel. The street frontage point is 12.5 meters from the front boundary of the parcel. Street frontage points are more suitable for routing applications.
Return8DecimalPlaceParcelLatLong	This option is available for Australia only. Specifies whether to return the original latitude and longitude, precise up to eight digits after the decimal. This is the latitude/longitude that the

optionName	Description / Valid Values
	candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This option is available only if you have the G-NAF database installed, and the database selected in the **Database** list on the Data tab includes the G-NAF database. This option only affects addresses matched to the G-NAF database. One of the following: **Y** Yes, return the original latitude/longitude, up to eight digits after the decimal. **N** No, do not return the original latitude/longitude.
CoordinateSystem	A coordinate system is a reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry. Spectrum™ Technology Platform supports systems recognized by the European Petroleum Survey Group (EPSG). Each country supports different coordinate systems. Depending on the country, you have one or more of the following options: **EPSG:4230** **EPSG:4283** Also known as the GDA94 coordinate system. **EPSG:4301** Also known as the Tokyo coordinate system. **EPSG:4326** Also known as the WGS84 coordinate system. **EPSG:27200** Also known as the NZGD49 coordinate system. **EPSG:27700** Also known as the British National Grid system.

Matching Options

Matching options let you set match restrictions, fallback, and multiple match settings so that the matching can be as strict or relaxed as you need. The strictest matching conditions require an exact match on house number, street name, postal code and no fallback to postal code centroids. The geocoder looks for an exact street address match within the postal code in the input address. Relaxing the conditions broadens the area in which it searches for a match. For example, by relaxing the postal code, the geocoder searches for candidates outside the postal code but within the city of your input address.

For guidelines on how to balance match rate and precision, see **Balancing Match Rate and Precision** on page 302.

Table 103: Matching Options

optionName	Description / Valid Values
KeepMultimatch	Specifies whether to return results when the address matches to multiple candidates in the database. If this option is not selected, an address that results in multiple candidates will fail to geocode. If you select this option, specify the maximum number of candidates to return using the MaxCandidates option (see below). One of the following: **Y** Yes, return candidates when multiple candidates are found. Default. **N** No, do not return candidates. Addresses that result in multiple candidates will fail to geocode.
MaxCandidates	If you specify KeepMultimatch=Y, this option specifies the maximum number of results to return. The default is 1.
ReturnRanges	Specifies whether to return address range information. If you enable this option, the output field `Ranges` will be included in the output. A range is a series of addresses along a street segment. For example, 5400-5499 Main St. is an address range representing addresses in the 5400 block of Main St. A range may represent just odd or even addresses within a segment, or both odd and even addresses. A range may also represent a single building with multiple units, such as an apartment building.
MaxRanges	If you choose to return ranges, this option specifies the maximum number of ranges to return for each candidate. Since the geocoder returns one candidate per segment, and since a segment may contain multiple ranges, this option allows you to see the other ranges in a candidate's segment.
MaxRangeUnits	If you choose to return ranges, this option specifies the maximum number of units (for example, apartments or suites) to return for each range. For example, if you were to geocode an office building at 65 Main St. containing four suites, there would be a maximum of four units returned for the building's range (65 Suite 1, 65 Suite 2, 65 Suite 3, and 65 Suite

optionName	Description / Valid Values
	4. If you were to specify a maximum number of units as 2, then only two units would be returned instead of all four.
ExpandCandidates	This option applies to U.K. addresses only. Specifies whether to return multiple close matches (one for each delivery point on the street). By default, if a street has multiple addresses/delivery points, GeocodeAddressGBR returns a single close match, with each individual delivery point available as a range.
CloseMatchesOnly	Specifies whether to return only those geocoded results that are close match candidates. For example, if there are 10 candidates and two of them are close candidates, and you enable this option, only the two close matching candidates would be returned instead of all 10. To specify what is considered a close match, use the **MustMatch** options. Address candidates are ranked according to how closely the input address matches these preferences. One of the following: **Y** Yes, return only close matches. **N** No, do not return only close matches. Default.
MatchMode	Specifies how to determine whether a candidate is a close match. One of the following: **CustomMode** This option allows you to specify which parts of a candidate address must match the input address to be considered a close match. Use the **MustMatch<Element>** options to specify the address elements you want. This is the default value for most countries. **ExactMode** All of a candidate address's elements must match in order for the candidate to be considered a close match. **CloseMode** Only the candidate address's house number must match in order for the candidate to be considered a close match. For Chile, China, Great Britain, Estonia, India, Indonesia, Latvia, Lithuania, Slovakia, Slovenia, Taiwan, and South Africa, only the street name and town must match. **RelaxedMode** All candidate addresses are considered a close match. This is the default value for Japan.

optionName	Description / Valid Values
MustMatchInput	Specifies whether candidates must match all non-blank input fields to be considered a close match. For example, if an input address contains a city and postal code, then candidates for this address must match the city and postal code to be considered a close match. One of the following: **Y** Yes, a candidate must match all input to be considered a close match. **N** No, a candidate does not have to match all input to be considered a close match. Default.
MustMatchHouseNumber	Specifies whether candidates must match the house number to be considered a close match. This option is not used for China, Indonesia, India, and Japan. If you select this option you should also require an exact match on street name. This option does not significantly affect performance. It does, however, affect the type of match if the candidate address corresponds to a segment that does not contain any ranges. The type of match can also be affected when the house number range for a candidate does not contain the input house number. One of the following: **Y** Yes, a candidate must match the house number to be considered a close match. **N** No, a candidate does not have to match the house number to be considered a close match.
MustMatchStreet	Specifies whether candidates must match the street name to be considered a close match. This option is not used for Japan. If a close match is found, the geocoder attempts expanded street name manipulation, which looks for candidates with names that sound like the input address or that are spelled improperly. For example, "Muller Strasse" for "Mueller Strasse". This slows down performance but increases the match rate . If the geocoding database is indexed, the performance impact is reduced. One of the following: **Y** Yes, a candidate must match the street name to be considered a close match.

optionName	Description / Valid Values
	N No, a candidate does not have to match the street name to be considered a close match.
MustMatchLocality	Specifies whether candidates must match the locality to be considered a close match. If you do not require exact matches on locality, the geocoder searches on the street address matched to the particular postal code, and considers other localities that do not match the name, but do match the postal code. The meaning of locality varies by country: • **Not used**—Australia (AUS), Austria (AUT), Belgium and Luxembourg (BEL), Chile (CHN), Germany (DEU), Denmark (DNK), Spain (ESP), Estonia (EST), Finland (FIN), France (FRA), Italy (ITA), Lithuania (LTU), Malaysia (MYS), The Netherlands (NLD), Norway (NOR), Portugal (PRT), Slovakia (SVK), Slovenia (SVN), Sweden (SWE), Thailand (THA) • **Locality**—Brazil (BRA), Switzerland (CHE), China (CHN), Czech Republic (CZE), The United Kingdom (GBR), Greece (GRC), Hong Kong (HKG), Hungary (HUN), India (IND), Ireland (IRL), Latvia (LVA), Macau (MAC), Mexico (MEX), Singapore (SGP), Turkey (TUR), Taiwan (TWN), South Africa (ZAF) • **Dissemination Area and Enumeration Area (DA and EA)**—Canada (CAN) • **Suburb**—New Zealand (NZL) • **Minor town, village, or locality**—Indonesia (IDN), Poland (POL) • **Neighborhood or barrio**—Argentina (ARG) • **City district (chome)**—Japan (JPN) This option is not available if you specify **GeocodeLevel=GeographicCentroid**. Specify one of the following values: **Y** Yes, a candidate must match the locality to be considered a close match. **N** No, a candidate does not have to match the locality to be considered a close match.
MustMatchCity	Specifies whether candidates must match the city to be considered a close match. For Japan, this field specifies whether the candidate must match the municipality subdivision (oaza). If you do not require exact matches on city, the geocoder searches on the street address matched

optionName	Description / Valid Values
	to the particular postal code, and considers other cities that do not match the name, but do match the postal code. One of the following: **Y** Yes, a candidate must match the city to be considered a close match. **N** No, a candidate does not have to match the city to be considered a close match. For Argentina, the city name is not always available in the source data. In those cases, the department (County field) is used for matching purposes and the department is returned in the City output field. Candidates matched in this manner are considered close matches.
MustMatchCounty	Specifies whether candidates must match the county to be considered a close match. The meaning of county varies by country: • ARG (Argentina)—Department • AUS (Australia)—Local Government Authority (LGA) • AUT (Austria)—Not used • BEL (Belgium)—Province • BRA (Brazil)—Not used • CAN (Canada)—Not used • CHE (Switzerland)—Province • CHL (Chile)—Not used • CHN (China)—Not used • CZE (Czech Republic)—District • DEU (Germany)—Kreis • DNK (Denmark)—Not used • ESP (Spain)—Province • EST (Estonia)—Not used • FIN (Finland)—Not used • FRA (France)—Department • GBR (Great Britain)—County • GRC (Greece)—District • HKG (Hong Kong)—Not used • HUN (Hungary)—Not used • IDN (Hungary)—SubDistrict • IND (India)—District • IRL (Ireland)—Province • ITA (Italy)—Province • JPN (Japan)—City (shi)

optionName	Description / Valid Values
	• LTU (Lithuania)—Not used • LVA (Latvia)—County • MAC (Macau)—TNot used • MEX (Mexico)—Not used • MYS (Malaysia)—Not used • NZL (New Zealand)—Not used • NLD (The Netherland)—Province • NOR (Norway)—Fylke • PHL (Philippines—District • POL (Poland)—District (poviat) • PRT (Portugal)—Not used • RUS (Russia)—District • SGP (Singapore)—Not used • SVK (Slovakia)—Not used • SVN (Slovenia)—Not used • SWE (Sweden)—Kommun • THA (Thailand)—Not used • TUR (Turkey)—District • TWN (Taiwan)—Not used • ZAF (South Africa—District One of the following: **Y** Yes, a candidate must match the county to be considered a close match. **N** No, a candidate does not have to match the county to be considered a close match.
MustMatchStateProvince	Specifies whether candidates must match the state or province to be considered a close match. The meaning of state/province varies by country: • ARG (Argentina)—Province or region • AUS (Australia)—State • AUT (Austria)—Not used • BEL (Belgium)—Not used • BRA (Brazil)—State • CAN (Canada)—Province • CHE (Switzerland)—Not used • CHL (Chile)—Not used • CHN (China)—Province

optionName	Description / Valid Values
	• CZE (Czech Republic)—Region • DEU (Germany)—Bundesland • DNK (Denmark)—Not used • ESP (Spain)—Not used • EST (Estonia)—Not used • FIN (Finland)—Not used • FRA (France)—Region • GRC (Greece)—Region • GBR (Great Britain)—Region • HKG (Hong Kong)—Province • HUN (Hungary)—County • IDN (Indonesia)—Not used • IND (India)—State • IRL (Ireland)—Not used • ITA (Italy)—Region • JPN (Japan)—Not used • LTU(Lithuania)—Not used • LVA (Latvia)—Not used • MAC (Macau)—State • MEX (Mexico)—Province • MYS (Malaysia)—State (negeri) • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Region • PHL (Philippines)—State • POL (Poland)—Province (voivodship) • PRT (Portugal)—Region • RUS (Russia)—Region • SGP (Singapore)—Region • SVK (Slovakia)—Not used • SVN (Slovenia)—Not used • SWE (Sweden)—Lan • THA (Thailand)—Province (changwat) • TUR (Turkey)—Province • TWN (Taiwan)—County or Special Municipality • ZAF (South Africa)—Not used One of the following: **Y** Yes, a candidate must match the state or province to be considered a close match.

optionName	Description / Valid Values
	N No, a candidate does not have to match the state or province to be considered a close match.
MustMatchPostalCode	Specifies whether candidates must match the postal code to be considered a close match. If you do not require exact match on postal codes, the geocoder searches a wider area for a match. While this results in slower performance, the match rate is higher because the request does not need to match exactly when it compares match candidates. This option is not available for Argentina, Japan, Hong Kong, Macau, Philippines, and South Africa. One of the following: **Y** Yes, a candidate must match the postal code to be considered a close match. **N** No, a candidate does not have to match the postal code to be considered a close match.

PreferPostalCodeOverCity	This option applies only to Australian addresses.
	
	Specifies whether to return a close match when the postal code and street match exactly but the city does not. If you enable this option, the City input field is ignored if the postal code and street address are an exact match. If you do not enable this option, there may be situations where street address and postal code input are an exact match but the city name is not, and close matches are not returned.
	
	For example, without this option enabled, the following input address would return no close matches:
	
	`5 East St`
	`GLENROY VIC 3046`
	
	With this option enabled, this same input address would return the following close match:
	
	`5 EAST ST`
	`HADFIELD VIC 3046`
	
	One of the following:
	
	Y Yes, return close matches if the postal code and street address are an exact match and ignore the city input.
	
	N No, do not return close matches if the postal code and street address are an exact match and the city is not. Default.

optionName	Description / Valid Values
MustMatchFullPostalCode	This option applies only to Canadian addresses. Specifies whether a candidate address must have a full postal code (FSA and LDU) to be considered a close match. Canadian postal codes are divided into two sections: the Forward Sortation Area (FSA) and the Local Delivery Unit (LDU). For example, the postal code M6H 2P8 has an FSA of M6H and an LDU of 2P8. Some candidate addresses may contain only the FSA. This option allows you to prevent such addresses from being classified a close match. One of the following: **Y** Yes, the candidate address must contain a full postal code to be considered a close match. **N** No, the candidate address does not have to contain a full postal code to be considered a close match. Default.
MustMatchPostalDistrict	This option applies only to U.K. addresses. Specifies whether the postal district portion of the postcode must match in order for the match to be considered a close match. UK postcodes are divided into two sections: the outward code, which is to the left of the space, and the inward code, which is to the right. The outward code represents the postal district. For example, in the postcode CB3 OHH, the postal district is CB3, which is Cambridge. If the city is not specified on input, the postcode sectory (outward code plus the first digit of inward code) is required for a close match.
MustMatch2DigitPostalCode	This option applies only to Spanish addresses. Specifies whether the first two digits of the postcode must match in order for the match to be considered a close match.
ReturnPOIWithStreet	This option applies only to Indian addresses. Specifies whether to return both point of interest (POI) matches and street matches when both street and POI information are provided in the input. If you select this option, POI information is returned with candidates that matched on street name. If you do not select this option, only street information is returned. One of the following: **Y** Yes, return POI information when both POI and street information are provided on input. Default.

<table>
<tr><th>optionName</th><th>Description / Valid Values</th></tr>
<tr><td></td><td>N No, do not return POI information when both POI and street information are provided on input.</td></tr>
<tr><td>Note: The India geocoder Prefer point of interest option is not functional, even though the selection is available. Ignore this entry.

PreferPOIOverStreet</td><td>Note: This selection will have no effect. A street match is always preferred for the India geocoder. Ignore this entry.

This option applies only to Indian addresses.

If you specify <code>ReturnPOIWithStreet=Y</code>, the default behavior is to list the street name matches first in the list of candidates (in other words, preferring the street matches). However you can choose to return POI candidates first using this option. Consider the following example:

FirmName: India Gate
AddressLine1: Lodi Road
City: Delhi

If POIs are returned and street names preferred (the default settings), the following candidates are returned in the order listed. Note that the street name candidates are at the top with the India Gate POI candidate (S8 result code) listed last. If POIs were preferred over street candidates, the India Gate candidate would be listed first.

<table>
<tr><th>StreetName</th><th>FirmName</th><th>PostalCode</th><th>Locality</th><th>City</th><th>ResultCode</th></tr>
<tr><td>Lodi Road</td><td></td><td>100003</td><td>Lodi Colony</td><td>Delhi</td><td>S4PNTSCA</td></tr>
<tr><td>Lodi Road</td><td></td><td>100013</td><td>Nizamuddin West</td><td>Delhi</td><td>S4PNTSCA</td></tr>
<tr><td>Lodi road</td><td>India Gate</td><td>100013</td><td>India Gate</td><td>Delhi</td><td>S8H—C-A</td></tr>
</table>
One of the following:

Y Yes, return POI candidates first in the list of candidates.

N No, do not return POI first in the list of candidates. Return street matches first. Default.</td></tr>
<tr><td>MustMatchPlaceName</td><td>This option applies only to Indian addresses.

Specifies whether a candidate address must match the place name to be considered a close match.

One of the following:

Y Yes, the candidate address must match the place name to be considered a close match.</td></tr>
</table>

optionName	Description / Valid Values	
	N	No, the candidate address does not have to match the place name to be considered a close match. Default.

Data Options

The Data tab allows you to specify which databases to use in geocoding. Databases contain the address and geocode data necessary to determine the geocode for a given address. There are two kinds of databases: standard databases and custom databases. Standard databases are those supplied by Pitney Bowes Software and based on address and geocoding data from postal authorities and suppliers of geographical data. Custom databases are databases you create to enhance or augment standard databases for your particular needs. For more information on custom databases, see **What Is a Custom Database?** on page 602.

For Australian geocoding, to achieve the best geocoding spatial precision use the G-NAF database. This provides point-level geocoding that places points within the land parcel boundary for a given address. The G-NAF database requires an additional license. Contact your sales representative for more information.

The following table lists the options available for specifying which databases to use and the search order of databases.

Table 104: Data Options

optionName	Description / Valid Values	
Database	Specifies the database to be used for geocoding. Only databases that have been defined in the Databases Resources panel in the Management Console are available.	
DatabasePreference	Specifies which geocoding databases to use. One of the following:	
	PreferCustom	Use both standard databases and custom databases, but give preference to candidates from custom databases. Use this option if you feel your custom database is superior to the standard database.
	PreferStandard	Use both standard databases and custom databases, but give preference to candidates from standard databases.
	CustomOnly	Use only custom databases. Ignore standard databases.
	StandardOnly	Use only standard databases. Ignore custom databases.

optionName	Description / Valid Values
	Both Use both standard databases and custom databases. In cases where candidates are returned from both, the standard database is preferred. Default. **Note:** This option is not available for Great Britain (GBR) because the GBR geocoder does not support custom databases. The results from a custom database have a "U" at the end of the result code. Results from an address database have an "A" at the end of the match score. For example: S5HPNTSCZA is a match score that comes from an address database, while S5HPNTSCZU comes from a custom database. For more information, see **Introduction** on page 597.
DatabaseSearchOrder	The name of one or more database resources to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. You can specify multiple database resources. If you specify more than one database, list them in order of preference. The order of the databases has an effect when there are close match candidates from different databases. The close matches that are returned come from the database that is first in the search list. Close matches from lower ranked databases are demoted to non-close matches. You can also use the order of the databases to perform fallback processing if you have an both an address point database and a street-level database installed for the country. List the address point database first and the street database second. If the address cannot be geocoded to the address point level, the geocoder will attempt to geocode it to the street level.

Output Data Options

The following table lists the options that control which data is returned in the output.

Table 105: Output Data Options

optionName	Description / Valid Values
ReturnOnlySimilarFirmNames	This option applies to the U.K. only. Specifies whether to return firm names only when the input firm name is similar to the firm name in the geocoding database. For example, if

optionName	Description / Valid Values
	the input firm name is "Pitney Bowes Business Insight" but the geocoding database returns "Pitney Bowes Software, Inc.", these two firm names are not similar. In most cases the input firm name must match the firm name in the database exactly. Some differences in abbreviations are considered similar enough to result in the firm name being returned. One of the following: **Y** Yes, return only firm names that are similar to the input firm name. **N** No, return firm names regardless of whether they are close to the input firm name. Default.

Output

The geocoder returns the latitude/longitude, standardized address, and result indicators. Result indicators describe how well the geocoder matched the input address to a known address and assigned a location; they also describe the overall status of a match attempt. If you are using the API, t T he output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section of the .

Address Output

The address may be identical to the input address if the input address was accurate, or it may be a standardized version of the input address, or it may be a candidate address when multiple matches are found.

Table 106: Address Output

columnName	Description
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
City	The municipality name. For Japan, the municipality subdivision (oaza).

columnName	Description
Country	The three-letter ISO 3166-1 Alpha 3 country code. Addresses from countries that do not have a dedicated geocoding stage contain the country code for the country of the geocoder. For example, Vatican City addresses have ITA in the Country field instead of VAT because Vatican City addresses are geocoded by GeocodeAddressITA. **AND (Andorra)**—Addresses from Andorra contain ESP (Spain) in the Country field. **GIB (Gibraltar)**—Addresses from Gibraltar contain ESP (Spain) in the Country field. **MCO (Monaco)**—Addresses from Monaco contain FRA (France) in the Country field. **SMR (San Marino)**—Addresses from San Marino contain ITA (Italy) in the Country field. **VAT (Vatican City)**—Addresses from Vatican City contain ITA (Italy) in the Country field.
County	The meaning of county varies by country: • ARG (Argentina)—Department • AUS (Australia)—The Local Government Authority (LGA) • AUT (Austria)—Not used • BEL (Belgium)—Province • BRA (Brazil)—Not used • CAN (Canada)—Not used • CHE (Switzerland)—Province • CHL (Chile)—District • CHN (China)—Not used • COL (Columbia)—TBD • CZE (Czech Republic)—District name or alias • DEU (Germany)—Kreis • DNK (Denmark)—Province • ESP (Spain)—Province • EST (Estonia)—District • FIN (Finland)—Not used • FRA (France)—Department • GBR (Great Britain)—Not used • GRC (Greece) —District • HKG (Hong Kong)—Not used • HUN (Hungary)—District • IDN (Indonesia)—Regency/City

columnName	Description
	• IND (India)—District • IRL (Ireland)—Province • ITA (Italy)—Province • JPN (Japan)—City (shi) • LTU (Lithuania)—County • LVA (Latvia)—District • MAC (Macau)—Not used • MEX (Mexico)—Province • MYS (Malaysia)—Not used • NLD (The Netherlands)—Province • NOR (Norway)—District (fylke/counties) • NZL (New Zealand)—Region • PHL (Philippines)—District • POL (Poland)—District (poviat) • PRT (Portugal)—Not used • RUS (Russia) —District • SGP (Singapore)—Not used • SVK (Slovakia)—Town • SVN (Slovenia)—Region • SWE (Sweden)—Kommun • THA (Thailand)—District (amphoe) • TUR (Turkey)—District • TWN (Taiwan)—Not used • ZAF (South Africa)—District **Note:** If you enter addresses with the Andorran country code (AND), only Andorran addresses are returned, including the AND country code. If you enter an Andorran with a country code of Spain (ESP), you will still get Andorran candidates, but with the ESP country code.
FirmName	Name of the company or a place name.
HouseNumber	The building number for the matched location. For Japan, this field contains the lot number.
HouseNumberHigh	The highest house number of the range in which the address resides.
HouseNumberLow	The lowest house number of the range in which the address resides.

columnName	Description
HouseNumberParity	Indicates if the house number range contains even or odd numbers or both. **E** Even **O** Odd **B** Both **U** Unknown
LastLine	Complete last address line (city, state/province, and postal code).
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
Locality	The meaning of locality varies by country: • ARG (Argentina)—Neighborhood or barrio • AUS (Australia)—Not used • AUT (Austria)—Not used • BEL (Belgium)—Not used • BRA (Brazil)—Locality • CAN (Canada)—Dissemination Area and Enumeration Area (DA and EA) • CHE (Switzerland)—Not used • CHL (Chile)—Locality • CHN (China)—Locality • COL (Columbia)—Locality TBD • CZE (Czech Republic)—Locality • DEU (Germany)—Not used • DNK (Denmark)—Not used • ESP (Spain)—Locality • EST (Estonia)—Locality • FIN (Finland)—Not used • FRA (France)—Not used • GBR (Great Britain)—Locality • GRC (Greece) —Locality • HKG (Hong Kong) —Locality • HUN (Hungary)—Locality • IDN (Indonesia)—Village • IND (India)—Locality • IRL (Ireland)—Not used

columnName	Description
	• ITA (Italy)—Locality • JPN (Japan)—City district (chome) • LTU (Lithuania)—Locality • LVA (Latvia)—Locality • MAC (Macau) —Locality • MEX (Mexico)—Locality • MYS (Malaysia)—Not used • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Suburb • PHL (Philippines)—Locality • POL (Poland)—Not used • PRT (Portugal)—Locality • RUS (Russia)—Locality • SGP (Singapore)—Not used • SVK (Slovakia)—Locality • SVN (Slovenia)—Locality • SWE (Sweden)—Not used • THA (Thailand)—Not used • TUR (Turkey)—Locality • TWN (Taiwan)—Locality • ZAF (South Africa)—Locality Geocode Address Global can geocode addresses in which the Locality is input without the City. For example, the following addresses are geocoded identically: calle nicaragua 4705, palermo, capital federal calle nicaragua 4705, palermo
NumberOfCandidateRanges	Indicates the number of ranges of which the candidate is a member. A candidate may be a part of multiple ranges if the candidate is a street instead of a building. To specify the number of ranges to return for each candidate, use the `MaxRanges` option.
NumberOfRangeUnits	Indicates the number of units included in the range. A unit is an address within a building, such as an apartment or office suite. To specify the number of units to return for each range, use the `MaxRangeUnits` option.
PostalCode	The postcode for the address. The format of the postcode varies by country.

columnName	Description
PostalCode.Addon	The second part of a postcode. For Canadian addresses this will be the LDU. This field is not used by most countries.
PreAddress	Miscellaneous information that appears before the street name.
PrivateMailbox	This field is not currently used.
Ranges	This is a list field containing the address ranges that exist on the street segment where the candidate address is located. A range is a series of addresses along a street segment. For example, 5400-5499 Main St. is an address range representing addresses in the 5400 block of Main St. A range may represent just odd or even addresses within a segment, or both odd and even addresses. A range may also represent a single building with multiple units, such as an apartment building. The Ranges field contains the following sub-fields: **Address** — This is a list filed that contains sub-fields for any address elements (AddressLine1, City, and so on) that are different from the candidate's address. **AdditionalFields** — A listing of country-specific information related to the address. The information contained in AdditionalFields varies by country. **HouseNumberHigh** — The highest address number for the range. **HouseNumberLow** — The lowest address number for the range. **SegmentParity** — Indicates the side of the street where the range is located. One of the following: **0** It is not known which side of the street the range is located on. **1** The range is on the left side of the street. **2** The range is on the right side of the street.

columnName	Description
	HouseNumberParity Indicates whether the range contains odd or even address numbers. One of the following: **0** The range contains both odd and even address numbers. **1** The range contains odd address numbers **2** The range contains even address numbers. **-1** It is not known whether the range contains odd or even house numbers.
	TotalRangeUnitsReturned The number of unit ranges returned for the address. A unit is an address within a building, such as an apartment or suite.
	RangeUnits A list of the ranges of units within the building. An example of units are apartments or suites. **Address** This is a list filed that contains sub-fields for any address elements (AddressLine1, City, and so on) that are different from the candidate's address. **UnitNumberHigh** The highest unit number. **UnitNumberLow** The lowest unit number.
SegmentCode	A unique ID that identifies a street segment. In Japan, this is the Jusho code. A Jusho Code is a point ID that represents a unique address.
SegmentParity	Indicates which side of the street has odd numbers. **L** Left side of the street **R** Right side of the street

columnName	Description
	B Both sides of the street **U** Undetermined
StateProvince	The meaning of StateProvince varies by country: • ARG (Argentina)—Region or province • AUS (Australia)—State • AUT (Austria)—Region • BEL (Belgium)—Not used • BRA (Brazil)State— • CAN (Canada)—Province • CHE (Switzerland)—Not used • CHL (Chile)—State • CHN (China)—Province • COL (Columbia)—State TBD • CZE (Czech Republic)—Region name or alias. For example, the region HLAVNÍ MESTO PRAHA can be aliased as Prag. • DEU (Germany)—Bundesland • DNK (Denmark)—Not used • ESP (Spain)—Region • EST (Estonia)—Not used • FIN (Finland)—Region (län) • FRA (France)—Region • GBR (Great Britain)—Region • GRC (Greece) —Region • HKG (Hong Kong) —Province • HUN (Hungary)—County • IDN (Indonesia)—Province • IND (India)—State • IRL (Ireland)—Not used • ITA (Italy)—Region • JPN (Japan)—Prefecture • LTU (Lithuania)—Not used • LVA (Latvia)—State • MAC (Macau) — State • MEX (Mexico)—Province • MYS (Malaysia)—State (negeri) • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Region

columnName	Description
	• PHL (Philippines)—State • POL (Poland)—Province (voivodship) • PRT (Portugal)—Region • RUS (Russia) —Region • SGP (Singapore)—Not used • SVK (Slovakia)—Not used • SVN (Slovenia)—Not used • SWE (Sweden)—Region (lan) • THA (Thailand)—Province (changwat) • TUR (Turkey)—Province • TWN (Taiwan)—County or Special Municipality • ZAF (South Africa)—State
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.
StreetName	The street name. In Japan, this contains the block. Japanese addresses typically do not have street names.
StreetPrefix	The type of street when the street type appears before the base street name. For example, AVENUE: 12 **AVENUE** B KALGOORLIE WA 6430
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 3rd St. N.
UnitNumberHigh	The highest unit number of the range in which the unit resides.
UnitNumberLow	The lowest unit number of the range in which the unit resides.

Geocode Output

Attention:
- GeocodeAddress[Country Code] (with the exception of GBR) is deprecated and may not be supported in future releases.
- Use Geocode Address Global for geocoding.

Table 107: Geocode Output

columnName	Description
CoordinateSystem	The coordinate system used to determine the latitude and longitude coordinates. A coordinate system specifies a map projection, coordinate units, etc. An example is EPSG:4326. EPSG stands for European Petroleum Survey Group.
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

Result Codes

Result codes contain information about the success or failure of the geocoding attempt, as well as information about the accuracy of the geocode.

Table 108: Result Code Output

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. For more information, see **Introduction** on page 597.
IsCloseMatch	Indicates whether or not the address is considered a close match. An address is considered close based on the "Close match criteria" options on the Matching tab. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
MultiMatchCount	For street address geocoding, the number of matching address positions found for the specified address.

columnName	Description
	For intersection geocoding, the number of matching street intersection positions found for the specified addresses.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	If the geocoder could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data • Multiple Matches Found • Exception occurred • Unable to initialize Geocoder • No Match Found
Status.Description	If the geocoder could not process the address, this field will show a description of the failure. **Problem + explanation** Returned when Status.Code = Internal System Error. **Geocoding Failed** Returned when Status.code = No Geocode Found. **No location returned** Returned when Status.code = No Geocode Found. **No Candidates Returned** The geocoder could not identify any candidate matches for the address. **Multiple Candidates Returned and Keep Multiple Matches not selected** The address resulted in multiple candidates. In order for the candidate address to be returned, you must specify `KeepMultimatch=Y`.
LocationPrecision	A code describing the precision of the geocode. One of the following: **0** No coordinate information is available for this candidate address. **1** Interpolated street address.

columnName	Description	
	2	Street segment midpoint.
	3	Postal code 1 centroid.
	4	Partial postal code 2 centroid.
	5	Postal code 2 centroid.
	6	Intersection.
	7	Point of interest.
	8	State/province centroid.
	9	County centroid.
	10	City centroid.
	11	Locality centroid.
	12	Additional point precision for unspecified custom item.
	13	Additional point precision for unspecified custom item.
	14	Additional point precision for unspecified custom item.
	15	Additional point precision for unspecified custom item.
	16	The result is an Address Point.
	17	The result was generated by using address point data to modify the candidates segment data.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.	

Country-Specific Output

The following topics describe output that's unique to specific countries.

Australia G-NAF Database Output

The following table lists output fields that are unique to the Australian Geocoded National Address File (G-NAF®) database. G-NAF is an optional database that is available for all six states and two territories. G-NAF is the only authoritative Australian national index of locality, street and number, validated with geographic coordinates.

Table 109: Austraila G-NAF Output

columnName	Description
AUS.GNAF_CONFIDENCE	A number indicating how many G-NAF datasets the address is found in. A higher confidence level means that the same address was found in more data contributor sources. One of the following: **<number>** The number of datasets the address was found in, minus 1. For example, a value of 0 indicates that the address was found in one contributor's dataset, a value of 1 indicates that the address was found in two contributors' datasets, a value of 2 indicates that the address was found in three contributors' datasets, and so forth. **-1** The address could not be found in any G-NAF dataset.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LATITUDE	The parcel latitude, precise to eight digits after the decimal. This is the latitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LONGITUDE	The parcel longitude, precise to eight digits after the decimal. This is the longitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_GEOCODE_LEVEL	A number indicating the level o f geocode for the address. Every principal address within the G-NAF database has at least a locality level geocode. They may also have a street level geocode and a point level geocode. One of the following: **0** No geocode.

columnName	Description
	1 Parcel level geocode only (no locality or street level geocode). **2** Street level geocode only (no locality or parcel level geocode). **3** Street and parcel level geocodes (no locality geocode). **4** Locality level geocode only (no street or parcel level geocode). **5** Locality and parcel level geocodes (no street level geocode). **6** Locality and street level geocodes (no parcel level geocodes). **7** Locality, street and parcel level geocodes.
AUS.GNAF_PID	The G-NAF Persistent Identifier (G-NAF PID) is a 14-character alphanumeric string that uniquely identifies each G-NAF address. The PID is constructed from a combination of the major address fields of the G-NAF Dictionary. An example of a G-NAF PID is: GAVIC411711441
AUS.GNAF_RELIABILITY	A number indicating the geocode precision. Reliability is related to the dictionary used to determine the geocode. Data with geocoded reliability levels 1, 2, and 3 is contained in the GNAF123 Dictionary. This is point (parcel) level geocoded data. Data with geocoded reliability levels 4, 5, and 6 is contained in the GNAF456 Dictionary. This contains non-parcel centroid geocoded data. **1** Geocode accuracy recorded to appropriate surveying standard. For example, this could apply to an address level geocode that was manually geocoded. Geocode resolution is sufficient to place the centroid within address site boundary with a GPS. **2** Geocode accuracy sufficient to place centroid within address site boundary. For example, this could apply to an address level geocode

columnName	Description
	that was automatically calculated as the centroid of the corresponding cadastre parcel. **3** Geocode accuracy sufficient to place centroid near (or possibly within) address site boundary. For example, this could apply to an address level geocode that was automatically calculated by calculating where on the road the address was likely to appear based upon other bounding geocoded addresses. **4** Geocode accuracy sufficient to associate address site with a unique road feature. For example, this could apply to a street level geocode that was automatically calculated by using the road centerline reference data. **5** Geocode resolution sufficient to associate address site with a unique locality or neighborhood. For example, this could apply to a locality level geocode that was automatically calculated as the centroid of the locality. **6** Geocode resolution sufficient to associate address site with a unique region. For example, this could apply to a locality level geocode that was derived from topographic feature.
AUS.LEVEL_NUMBER	The number of a floor or level in a multi-story building. For example, **Floor 2** , 17 Jones Street The G-NAF database includes level information for some Australian states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include

columnName	Description
	level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.LEVEL_TYPE	The label used for a floor of a multi-story building. For example, "Level" or "Floor". In this example, the level type is "Level": Suite 3 **Level 7**, 17 Jones Street In this example, Suite 3 is a unit. The G-NAF database includes level information for some Australian states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.MESH_BLOCK_ID	A Meshblock is the smallest geographic unit for which statistical data is collected by the Australian Bureau of Statistics (ABS). Meshblocks usually contain a minimum of 20 to 50 households. This is about one fifth the size of a Collection District (CD). You can use the Meshblock ID to do additional attributions against your own data.

Canada-Specific Output

The following table lists output fields that are unique to Canada.

Table 110: Canada-Specific Output

columnName	Description
CAN.Census_CD	The Census Division (CD) in which the address is located. For more information, see the **Census Division definition** available on the Statistics Canada website.
CAN.Census_CMA	The Census Metropolitan Area (CMA) in which the address is located. For more information, see the **Census Metropolitan Area definition** on the Statistics Canada website.
CAN.Census_CSD	The Census Subdivision (CSD) in which the address is located. For more information, see the **Census Subdivision definition** on the Statistics Canada website.
CAN.Census_CT	The Census Tract (CT) in which the address is located. For more information, see the **Census Tract definition** on the Statistics Canada website.
CAN.Census_DA	The Dissemination Area (DA) in which the address is located. For more information, see the **Dissemination Area definition** available on the Statistics Canada website.

New Zealand Output

The following table lists output fields that are unique to New Zealand.

Table 111: New Zealand-Specific Output

columnName	Description
NZL.NZL_MESHBLOCK_ID	A Meshblock is the smallest geographic unit for which statistical data is collected by Statistics New Zealand. Meshblocks vary in size from part of a city block to large areas of rural land.
NZL.ALIASED_SUBURB	An alternative to the officially-recognized suburb name.

GeocodeAddressWorld

The GeocodeAddressWorld component takes an address located in any of the supported countries and returns the city centroid or, for some countries, postal centroid. GeocodeAddressWorld can geocode to the city centroid level, and for some counties the postal centroid level. It cannot geocode to the street address level. If you require address-level geocoding, use GeocodeAddressGlobal.

GeocodeAddressWorld is typically used as a fallback geocoder to cover countries for which a Geocode Address Global country is not available. For example, you may have licensed the Australia geocoder because you are primarily interested in geocoding Australian addresses. However, your data may have some records with locations outside Australia. In this case you could use GeocodeAddressWorld to provide centroid geocodes for locations outside Australia, while using the Australia geocoders to provide more precise geocodes for Australian addresses.

The GeocodeAddressWorld component is an optional part of the Enterprise Geocoding Module. For more information about Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Geocode Precision

GeocodeAddressWorld automatically provides the best geocode possible based on the data you provide on input. If you provide a city and valid postal code, you will receive a postal code centroid. If you provide a city and an invalid postal code, or a city and no postal code, GeocodeAddressWorld will automatically fall back to the geographic centroid, providing the centroid for the city. The postal code returned in this case will be the postal code of the city centroid.

Postal Geocoding

Geocode Address World can geocode to a postal centroid if postcode information is available from the country. Postcode information can come from any of the data sources (TomTom, GeoNames, or Pitney Bowes). See **Country Postal Data Coverage** on page 443 for a summary of Geocode Address World postal data coverage. Depending on the country, postal geocoding may provide more accurate results that geographic geocoding.

Postal level geocoding is possible if these conditions are met:

- Your input address consists of a valid postcode.
- The data source contains postcode information for the country. Not every country has postcode data.

Geocode Address World may return multiple close matches for postal geocoding. For example, a postcode of 12180 matches Troy NY but the identical postcode occurs in several other countries. If the input is the postcode only, then all those candidates are returned as close matches.

However, if the input includes geographic address elements (such as country, state, region, or city name), Geocode Address World may be able to use that information to return a more accurate single close match. If you want to use geographic address content to refine your postal geocoding results, consider the following:

- Different countries derive their postal data from either the TomTom, GeoNames, or Pitney Bowes sources. Therefore, the available geographic content in the postal data source varies by country. For example, city name (City) is a close match weighting factor for countries that use the GeoNames postal data source, but city name is ignored for countries that use the TomTom postal data source. See **Data**

Sources and Coverage on page 429 for information on the geographic content of the TomTom, GeoNames, and Pitney Bowes data sources.

Postal Geocoding with Geographic Information

In this postal geocoding example, the input address includes a valid postcode of 41012 and the province (StateProvince) of Emilia Romagna. A street address is provided, but this is ignored for postal geocoding.

Fornaci 40
Emilia Romagna
41012

Because the TomTom postal data source for Italy includes StateProvince, the province of Emilia Romagna is considered when evaluating close matches. Therefore, Emilia Romagna, Italy with the matching 41012 postal code is returned as the single close match with a Z1 result code. Candidates with a 41012 postcode from other countries are returned as non-close candidates. If StateProvince or Country information was not provided on input, then Geocode Address World would return multiple close matches because the five-digit 41012 postcode can be found in a number of countries.

Note: The geographic content must be present in the postal data source in order refine postal geocoding results. For example, the Italy TomTom postal data source does not include city/town (City). So if you input the city of Carpi with the 41012 postal code, Geocode Address World ignores the city name and returns multiple close matches for the 41012 postal code (unless you also specified the ITA country name). See **Data Sources and Coverage** on page 429 for information on the geographic content of the TomTom, GeoNames, and Pitney Bowes data sources.

Geographic Geocoding

Geocode Address World can geocode to the centroid of an administrative division (such as town or village).

Geocode Address World can geocode to the geographic level if these conditions are met:

- Your input addresses contains accurate geographic information without valid postcode address content in the input. If the address in question includes valid postcode input, then Geocode Address World will attempt postal geocoding.
- The data source contains geographic level information for the country. Geographic information can come from any of the data sources (TomTom, GeoNames, or Pitney Bowes).
- Country name or country ISO codes are not required, but if included, they must be matched. Including the country name may produce better close matches.

Geographic Geocoding to City

In this example, the input address includes the city (City) of Vaihingen an der Enz. The country is not specified in this example. The street address information (street name and number) is ignored for the purposed of geographic geocoding.

Muldenweg 2
Vaihingen an der Enz

Geocode Address World returns a G3 close match candidate. Even though the country was not specified, Geocode Address World identifies one close match in Germany (DEU).

StateProvince: Baden-Württemberg
County: Ludwigsburg
City: Vaihingen an der Enz
Country: DEU
Result Code: G3
X: 8.95948
Y: 48.930059

Geographic Geocoding with Common City Name

In this example, the input address includes the city (City) of Venice. This city name occurs in a number of countries, but the country is not specified on input.

St Marks Plaza
Venice

Geocode Address World selects Venice, Italy as the close match candidate because of its large population (approximately 270,000) and because Venice is the administrative capital of the Veneto region of Italy. A number of non-close matches may also be returned for cities of Venice in other countries. The close match candidate for Venice, ITA is:

StateProvince: Veneto
County: Venezia
City: Venice
Country: ITA
Result Code: G3
X: 12.33878
Y: 45.43434

Geographic Geocoding with State/Province Abbreviation

In this example the input address includes the city name of Rome and GA, which is the abbreviation for the state of Georgia in the USA. See **State or Province Abbreviations** on page 454 to see the countries for which state/province abbreviations are recognized. Because the state abbreviation is used, it is not necessary to specify the country name.

Rome, GA

Geocode Address World considers the StateProvince and returns a close match for Rome, Georgia USA. Even though Rome, Italy is a much larger city and is the

capital of Italy, that is returned as a non-close candidate because the StateProvince (GA) that was specified on input

StateProvince: Georgia
County: Floyd
City: Rome
Country: USA
Result Code: G3
X: -85.16467
Y: 34.25704

Geographic Geocoding to Locality

In this example, the input address includes the locality of Altamira and province abbreviation of GRO Geocode Address World recognizes the state abbreviation of GRO, so the country name is not necessary.

City: Altamira
StateProvince: GRO

In this example, Geocode Address World returns a close match to Locality) of Altamira even if Altamira was input as City. The (StateProvince) of GRO is also returned. If Guerrero is entered as StateProvince then Guerrero is returned.

StateProvince: GRO
City: ACAPULCO DE JUÁREZ
Locality: ALTAMIRA
Country: MEX
Result Code: G4
X: 99.87984
Y: 16.87637

Address input can be formatted into separate input fields or input can be unformatted (single line input). Geocoding of unformatted input is shown in **Single Line Input** on page 455.

Data Sources and Coverage

Geocode Address World relies on several data sources to build its comprehensive worldwide address databases. If an input address cannot be located using one of these data sources, then Geocode Address World uses one of the other data sources. The best available candidate is returned.

These data sources (for both geographic and postal data) are used in the listed order:

- TomTom data
- GeoNames data
- Pitney Bowes World data

Geocode Address World is partitioned into six databases based on continent. The geographic and postal data is integrated into each address dictionary to support both geographic geocoding and postal geocoding.

- Africa
- Asia
- Europe
- NorthAmerica
- Oceana
- SouthAmerica

See **Geographic Geocoding** on page 427 for a description and examples of geographic geocoding. See **Postal Geocoding** on page 426 for a description and examples of postal geocoding.

The postal source data can access the geographic content, which can be used to refine postal results. That is, geographic information (country name and administrative divisions) can be used to help evaluate close matches when the same postal code can be found in different countries.

Depending on the source of the postal data, the following geographic information is available to help refine postal results:

- TomTom source: Country, StateProvince
- GeoNames source: Country, StateProvince and City
- Pitney Bowes World source: Country, StateProvince, County, City, and Locality

Note: The Geocode Address World data set contains data licensed from the GeoNames Project (**http://www.geonames.org**) provided under the Creative Commons Attribution License ("Attribution License") located at **http://creativecommons.org/licenses/by/3.0/legalcode**. Your use of the GeoNames data (described in the Spectrum User Manual) is governed by the terms of the Attribution License, and any conflict between your agreement with PBSI and the Attribution License will be resolved in favor of the Attribution License solely as it relates to your use of the GeoNames data.

Country Coverage

Geocode Address World includes coverage for almost every country in the world. The accuracy and scope of coverage varies depending on the quality of the available data source. Some countries include postcode data, while other countries have geographic coverage only.

See **Data Sources and Coverage** on page 429 for more information on the TomTom, GeoNames, and Pitney Bowes geographic and postal data sources.

For a complete list of Geographic coverage by country, see **Country Geographic Data Coverage** on page 430. For Postal coverage by country, see **Country Postal Data Coverage** on page 443.

Country Geographic Data Coverage

Table 112: Country Names and Geographic Data Coverage

Country Name	ISO 3166 Country Code	Data Source	Vintage
AFGHANISTAN	AFG	GeoNames	2011.07
ALAND ISLANDS	ALA	GeoNames	2011.07
ALBANIA	ALB	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
ALGERIA	DZA	GeoNames	2011.07
AMERICAN SAMOA	ASM	GeoNames	2011.07
ANDORRA	AND	TomTom	2011.06
ANGOLA	AGO	TomTom	2011.06
ANGUILLA	AIA	GeoNames	2011.07
ANTARCTICA	ATA	GeoNames	2011.07
ANTIGUA AND BARBUDA	ATG	GeoNames	2011.07
ARGENTINA	ARG	TomTom	2011.06
ARMENIA	ARM	GeoNames	2011.07
ARUBA	ABW	GeoNames	2011.07
AUSTRALIA	AUS	GeoNames	2011.07
AUSTRIA	AUT	TomTom	2011.06
AZERBAIJAN	AZE	GeoNames	2011.07
BAHAMAS	BHS	GeoNames	2011.07
BAHRAIN	BHR	TomTom	2011.06
BANGLADESH	BGD	GeoNames	2011.07
BARBADOS	BRB	GeoNames	2011.07
BELARUS	BLR	TomTom	2011.06
BELGIUM	BEL	TomTom	2011.06
BELIZE	BLZ	GeoNames	2011.07
BENIN	BEN	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
BERMUDA	BMU	GeoNames	2011.07
BHUTAN	BTN	GeoNames	2011.07
BOLIVIA	BOL	GeoNames	2011.07
BONAIRE, SINT EUSTATIUS AND SABA	BES	GeoNames	2011.07
BOSNIA AND HERZEGOWINA	BIH	TomTom	2011.06
BOTSWANA	BWA	TomTom	2011.06
BOUVET ISLAND	BVT	GeoNames	2011.07
BRAZIL	BRA	TomTom	2011.06
BRITISH INDIAN OCEAN TERRITORY	IOT	GeoNames	2011.07
BRUNEI DARUSSALAM	BRN	TomTom	2011.06
BULGARIA	BGR	TomTom	2011.06
BURKINA FASO	BFA	TomTom	2011.06
BURUNDI	BDI	GeoNames	2011.07
CAMBODIA	KHM	GeoNames	2011.07
CAMEROON	CMR	TomTom	2011.06
CANADA	CAN	TomTom	2011.06
CAPE VERDE	CPV	GeoNames	2011.07
CAYMAN ISLANDS	CYM	GeoNames	2011.07
CENTRAL AFRICAN REPUBLIC	CAF	GeoNames	2011.07
CHAD	TCD	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
CHILE	CHL	TomTom	2011.06
CHINA	CHN	GeoNames	2011.07
CHRISTMAS ISLAND	CXR	GeoNames	2011.07
COCOS (KEELING) ISLANDS	CCK	GeoNames	2011.07
COLOMBIA	COL	GeoNames	2011.07
COMOROS	COM	GeoNames	2011.07
CONGO	COG	TomTom	2011.06
CONGO, DEMOCRATIC REPUBLIC OF THE	COD	TomTom	2011.06
COOK ISLANDS	COK	GeoNames	2011.07
COSTA RICA	CRI	GeoNames	2011.07
COTE D'IVOIRE	CIV	GeoNames	2011.07
CROATIA (LOCAL NAME: HRVATSKA)	HRV	TomTom	2011.06
CUBA	CUB	GeoNames	2011.07
CURAÇAO	CUW	GeoNames	2011.07
CYPRUS	CYP	GeoNames	2011.07
CZECH REPUBLIC	CZE	TomTom	2011.06
DENMARK	DNK	GeoNames	2011.07
DJIBOUTI	DJI	GeoNames	2011.07
DOMINICA	DMA	GeoNames	2011.07
DOMINICAN REPUBLIC	DOM	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
ECUADOR	ECU	GeoNames	2011.07
EGYPT	EGY	TomTom	2011.06
EL SALVADOR	SLV	GeoNames	2011.07
EQUATORIAL GUINEA	GNQ	GeoNames	2011.07
ERITREA	ERI	GeoNames	2011.07
ESTONIA	EST	TomTom	2011.06
ETHIOPIA	ETH	GeoNames	2011.07
FALKLAND ISLANDS (MALVINAS)	FLK	GeoNames	2011.07
FAROE ISLANDS	FRO	GeoNames	2011.07
FIJI	FJI	GeoNames	2011.07
FINLAND	FIN	TomTom	2011.06
FRANCE	FRA	TomTom	2011.06
FRENCH GUIANA	GUF	TomTom	2011.06
FRENCH POLYNESIA	PYF	GeoNames	2011.07
FRENCH SOUTHERN TERRITORIES	ATF	GeoNames	2011.07
GABON	GAB	TomTom	2011.06
GAMBIA	GMB	GeoNames	2011.07
GEORGIA	GEO	GeoNames	2011.07
GERMANY	DEU	TomTom	2011.06
GHANA	GHA	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
GIBRALTAR	GIB	GeoNames	2011.07
GREECE	GRC	TomTom	2011.06
GREENLAND	GRL	GeoNames	2011.07
GRENADA	GRD	GeoNames	2011.07
GUADELOUPE	GLP	TomTom	2011.06
GUAM	GUM	GeoNames	2011.07
GUATEMALA	GTM	GeoNames	2011.07
GUERNSEY	GGY	GeoNames	2011.07
GUINEA	GIN	GeoNames	2011.07
GUINEA-BISSAU	GNB	GeoNames	2011.07
GUYANA	GUY	GeoNames	2011.07
HAITI	HTI	GeoNames	2011.07
HEARD AND MCDONALD ISLANDS	HMD	GeoNames	2011.07
HONDURAS	HND	GeoNames	2011.07
HONG KONG	HKG	TomTom	2011.06
HUNGARY	HUN	TomTom	2011.06
ICELAND	ISL	GeoNames	2011.07
INDIA	IND	GeoNames	2011.07
INDONESIA	IDN	TomTom	2011.06
IRAN (ISLAMIC REPUBLIC OF)	IRN	GeoNames	2011.07
IRAQ	IRQ	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
IRELAND	IRL	TomTom	2011.06
ISLE OF MAN	IMN	GeoNames	2011.07
ISRAEL	ISR	GeoNames	2011.07
ITALY	ITA	TomTom	2011.06
JAMAICA	JAM	GeoNames	2011.07
JAPAN	JPN	GeoNames	2011.07
JERSEY	JEY	GeoNames	2011.07
JORDAN	JOR	GeoNames	2011.07
KAZAKHSTAN	KAZ	GeoNames	2011.07
KENYA	KEN	TomTom	2011.06
KIRIBATI	KIR	GeoNames	2011.07
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	PRK	GeoNames	2011.07
KOREA, REPUBLIC OF	KOR	GeoNames	2011.07
KUWAIT	KWT	TomTom	2011.06
KYRGYZSTAN	KGZ	GeoNames	2011.07
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LAO	GeoNames	2011.07
LATVIA	LVA	TomTom	2011.06
LEBANON	LBN	GeoNames	2011.07
LESOTHO	LSO	TomTom	2011.06
LIBERIA	LBR	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
LIBYAN ARAB JAMAHIRIYA	LBY	GeoNames	2011.07
LIECHTENSTEIN	LIE	GeoNames	2011.07
LITHUANIA	LTU	TomTom	2011.06
LUXEMBOURG	LUX	TomTom	2011.06
MACAO	MAC	TomTom	2011.06
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MKD	TomTom	2011.06
MADAGASCAR	MDG	GeoNames	2011.07
MALAWI	MWI	TomTom	2011.06
MALAYSIA	MYS	TomTom	2011.06
MALDIVES	MDV	GeoNames	2011.07
MALI	MLI	TomTom	2011.06
MALTA	MLT	TomTom	2011.06
MARSHALL ISLANDS	MHL	GeoNames	2011.07
MARTINIQUE	MTQ	GeoNames	2011.07
MAURITANIA	MRT	TomTom	2011.06
MAURITIUS	MUS	TomTom	2011.06
MAYOTTE	MYT	GeoNames	2011.07
MEXICO	MEX	TomTom	2011.06
MICRONESIA, FEDERATED STATES OF	FSM	GeoNames	2011.07
MOLDOVA, REPUBLIC OF	MDA	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
MONACO	MCO	GeoNames	2011.07
MONGOLIA	MNG	GeoNames	2011.07
MONTENEGRO	MNE	TomTom	2011.06
MONTSERRAT	MSR	GeoNames	2011.07
MOROCCO	MAR	TomTom	2011.06
MOZAMBIQUE	MOZ	TomTom	2011.06
MYANMAR	MMR	GeoNames	2011.07
NAMIBIA	NAM	TomTom	2011.06
NAURU	NRU	GeoNames	2011.07
NEPAL	NPL	GeoNames	2011.07
NETHERLANDS	NLD	TomTom	2011.06
NETHERLANDS ANTILLES	ANT	Pitney Bowes	C. 2006
NEW CALEDONIA	NCL	GeoNames	2011.07
NEW ZEALAND	NZL	GeoNames	2011.07
NICARAGUA	NIC	GeoNames	2011.07
NIGER	NER	TomTom	2011.06
NIGERIA	NGA	TomTom	2011.06
NIUE	NIU	GeoNames	2011.07
NORFOLK ISLAND	NFK	GeoNames	2011.07
NORTHERN MARIANA ISLANDS	MNP	GeoNames	2011.07
NORWAY	NOR	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
OMAN	OMN	TomTom	2011.06
PAKISTAN	PAK	GeoNames	2011.07
PALAU	PLW	GeoNames	2011.07
PALESTINIAN TERRITORY, OCCUPIED	PSE	GeoNames	2011.07
PANAMA	PAN	GeoNames	2011.07
PAPUA NEW GUINEA	PNG	GeoNames	2011.07
PARAGUAY	PRY	GeoNames	2011.07
PERU	PER	GeoNames	2011.07
PHILIPPINES	PHL	TomTom	2011.06
PITCAIRN	PCN	GeoNames	2011.07
POLAND	POL	TomTom	2011.06
PORTUGAL	PRT	TomTom	2011.06
PUERTO RICO	PRI	GeoNames	2011.07
QATAR	QAT	TomTom	2011.06
REUNION	REU	TomTom	2011.06
ROMANIA	ROU	TomTom	2011.06
RUSSIAN FEDERATION	RUS	TomTom	2011.06
RWANDA	RWA	GeoNames	2011.07
SAINT BARTHÉLEMY	BLM	GeoNames	2011.07
SAINT HELENA, ASCENSION AND TRISTAN DA CUNHA	SHN	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
SAINT KITTS AND NEVIS	KNA	GeoNames	2011.07
SAINT LUCIA	LCA	GeoNames	2011.07
SAINT MARTIN (FRENCH PART	MAF	GeoNames	2011.07
SAINT PIERRE AND MIQUELON	SPM	GeoNames	2011.07
SAINT VINCENT AND THE GRENADINES	VCT	GeoNames	2011.07
SAMOA	WSM	GeoNames	2011.07
SAN MARINO	SMR	TomTom	2011.06
SAO TOME AND PRINCIPE	STP	GeoNames	2011.07
SAUDI ARABIA	SAU	TomTom	2011.06
SENEGAL	SEN	TomTom	2011.06
SERBIA	SRB	TomTom	2011.06
SEYCHELLES	SYC	GeoNames	2011.07
SIERRA LEONE	SLE	GeoNames	2011.07
SINGAPORE	SGP	TomTom	2011.06
SINT MAARTEN (DUTCH PART)	SXM	GeoNames	2011.07
SLOVAKIA (SLOVAK REPUBLIC)	SVK	TomTom	2011.06
SLOVENIA	SVN	TomTom	2011.06
SOLOMON ISLANDS	SLB	GeoNames	2011.07
SOMALIA	SOM	GeoNames	2011.07
SOUTH AFRICA	ZAF	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	SGS	GeoNames	2011.07
SPAIN	ESP	TomTom	2011.06
SRI LANKA	LKA	GeoNames	2011.07
SUDAN	SDN	GeoNames	2011.07
SURINAME	SUR	GeoNames	2011.07
SVALBARD AND JAN MAYEN ISLANDS	SJM	GeoNames	2011.07
SWAZILAND	SWZ	TomTom	2011.06
SWEDEN	SWE	TomTom	2011.06
SWITZERLAND	CHE	TomTom	2011.06
SYRIAN ARAB REPUBLIC	SYR	GeoNames	2011.07
TAIWAN, PROVINCE OF CHINA	TWN	TomTom	2011.06
TAJIKISTAN	TJK	GeoNames	2011.07
TANZANIA, UNITED REPUBLIC OF	TZA	TomTom	2011.06
THAILAND	THA	TomTom	2011.06
TIMOR-LESTE	TLS	GeoNames	2011.07
TOGO	TGO	TomTom	2011.06
TOKELAU	TKL	GeoNames	2011.07
TONGA	TON	GeoNames	2011.07
TRINIDAD AND TOBAGO	TTO	GeoNames	2011.07
TUNISIA	TUN	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
TURKEY	TUR	TomTom	2011.06
TURKMENISTAN	TKM	GeoNames	2011.07
TURKS AND CAICOS ISLANDS	TCA	GeoNames	2011.07
TUVALU	TUV	GeoNames	2011.07
UGANDA	UGA	TomTom	2011.06
UKRAINE	UKR	TomTom	2011.06
UNITED ARAB EMIRATES	ARE	TomTom	2011.06
UNITED KINGDOM	GBR	TomTom	2011.06
UNITED STATES	USA	GeoNames	2011.07
UNITED STATES MINOR OUTLYING ISLANDS	UMI	GeoNames	2011.07
URUGUAY	URY	TomTom	2011.06
UZBEKISTAN	UZB	GeoNames	2011.07
VANUATU	VUT	GeoNames	2011.07
VATICAN CITY STATE (HOLY SEE)	VAT	GeoNames	2011.07
VENEZUELA	VEN	GeoNames	2011.07
VIET NAM	VNM	GeoNames	2011.07
VIRGIN ISLANDS (BRITISH)	VGB	GeoNames	2011.07
VIRGIN ISLANDS (U.S.)	VIR	GeoNames	2011.07
WALLIS AND FUTUNA ISLANDS	WLF	GeoNames	2011.07
WESTERN SAHARA	ESH	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
YEMEN	YEM	GeoNames	2011.07
ZAMBIA	ZMB	TomTom	2011.06
ZIMBABWE	ZWE	TomTom	2011.06

Country Postal Data Coverage

Table 113: Country Names and Postal Data Coverage

Country Name	ISO 3166 Country Code	Data Source	Vintage
ALGERIA	DZA	Pitney Bowes	C. 2006
AMERICAN SAMOA	ASM	GeoNames	2011.07
ANDORRA	AND	TomTom	2011.06
ARGENTINA	ARG	GeoNames	2011.07
ARMENIA	ARM	Pitney Bowes	C. 2006
AUSTRALIA	AUS	GeoNames	2011.07
AUSTRIA	AUT	TomTom	2011.06
AZERBAIJAN	AZE	Pitney Bowes	C. 2006
BAHRAIN	BHR	Pitney Bowes	C. 2006
BANGLADESH	BGD	GeoNames	2011.07
BELARUS	BLR	Pitney Bowes	C. 2006
BELGIUM	BEL	TomTom	2011.06
BERMUDA	BMU	Pitney Bowes	C. 2006
BOSNIA AND HERZEGOWINA	BIH	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
BRAZIL	BRA	TomTom	2011.09
BRITISH INDIAN OCEAN TERRITORY	IOT	Pitney Bowes	C. 2006
BRUNEI DARUSSALAM	BRN	Pitney Bowes	C. 2006
BULGARIA	BGR	GeoNames	2011.07
CAMBODIA	KHM	Pitney Bowes	C. 2006
CANADA	CAN	TomTom	2011.09
CAPE VERDE	CPV	Pitney Bowes	C. 2006
CHILE	CHL	Pitney Bowes	C. 2006
CHINA	CHN	Pitney Bowes	C. 2006
CHRISTMAS ISLAND	CXR	Pitney Bowes	C. 2006
COCOS (KEELING) ISLANDS	CCK	Pitney Bowes	C. 2006
COSTA RICA	CRI	Pitney Bowes	C. 2006
CROATIA (LOCAL NAME: HRVATSKA)	HRV	GeoNames	2011.07
CUBA	CUB	Pitney Bowes	C. 2006
CYPRUS	CYP	Pitney Bowes	C. 2006
CZECH REPUBLIC	CZE	TomTom	2011.06
DENMARK	DNK	GeoNames	2011.07
DOMINICAN REPUBLIC	DOM	GeoNames	2011.07
ECUADOR	ECU	Pitney Bowes	C. 2006
EGYPT	EGY	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
EL SALVADOR	SLV	Pitney Bowes	C. 2006
ESTONIA	EST	TomTom	2011.06
ETHIOPIA	ETH	Pitney Bowes	C. 2006
FALKLAND ISLANDS (MALVINAS)	FLK	Pitney Bowes	C. 2006
FAROE ISLANDS	FRO	GeoNames	2011.07
FINLAND	FIN	TomTom	2011.06
FRANCE	FRA	TomTom	2011.06
FRENCH GUIANA	GUF	GeoNames	2011.07
FRENCH POLYNESIA	PYF	Pitney Bowes	C. 2006
GEORGIA	GEO	Pitney Bowes	C. 2006
GERMANY	DEU	TomTom	2011.06
GREECE	GRC	TomTom	2011.06
GREENLAND	GRL	GeoNames	2011.07
GUADELOUPE	GLP	GeoNames	2011.07
GUAM	GUM	GeoNames	2011.07
GUATEMALA	GTM	GeoNames	2011.07
GUERNSEY	GGY	GeoNames	2011.07
GUINEA	GIN	Pitney Bowes	C. 2006
GUINEA-BISSAU	GNB	Pitney Bowes	C. 2006
HAITI	HTI	Pitney Bowes	C. 2006
HONDURAS	HND	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
HUNGARY	HUN	GeoNames	2011.07
ICELAND	ISL	GeoNames	2011.07
INDIA	IND	GeoNames	2011.07
INDONESIA	IDN	TomTom	2011.06
IRAN (ISLAMIC REPUBLIC OF)	IRN	Pitney Bowes	C. 2006
IRAQ	IRQ	Pitney Bowes	C. 2006
IRELAND	IRL	Pitney Bowes	C. 2006
ISLE OF MAN	IMN	GeoNames	2011.07
ISRAEL	ISR	Pitney Bowes	C. 2006
ITALY	ITA	TomTom	2011.06
JAMAICA	JAM	Pitney Bowes	C. 2006
JAPAN	JPN	GeoNames	2011.07
JERSEY	JEY	GeoNames	2011.07
JORDAN	JOR	Pitney Bowes	C. 2006
KAZAKHSTAN	KAZ	Pitney Bowes	C. 2006
KENYA	KEN	Pitney Bowes	C. 2006
KOREA, REPUBLIC OF	KOR	Pitney Bowes	C. 2006
KUWAIT	KWT	Pitney Bowes	C. 2006
KYRGYZSTAN	KGZ	Pitney Bowes	C. 2006
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LAO	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
LATVIA	LVA	TomTom	2011.06
LEBANON	LBN	Pitney Bowes	C. 2006
LESOTHO	LSO	Pitney Bowes	C. 2006
LIBERIA	LBR	Pitney Bowes	C. 2006
LIECHTENSTEIN	LIE	GeoNames	2011.07
LITHUANIA	LTU	TomTom	2011.06
LUXEMBOURG	LUX	GeoNames	2011.07
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MKD	GeoNames	2011.07
MADAGASCAR	MDG	Pitney Bowes	C. 2006
MALAYSIA	MYS	GeoNames	2011.07
MALDIVES	MDV	Pitney Bowes	C. 2006
MALTA	MLT	Pitney Bowes	C. 2006
MARSHALL ISLANDS	MHL	GeoNames	2011.07
MARTINIQUE	MTQ	GeoNames	2011.07
MAYOTTE	MYT	GeoNames	2011.07
MEXICO	MEX	TomTom	2011.06
MICRONESIA, FEDERATED STATES OF	FSM	Pitney Bowes	C. 2006
MOLDOVA, REPUBLIC OF	MDA	GeoNames	2011.07
MONACO	MCO	GeoNames	2011.07
MONGOLIA	MNG	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
MOROCCO	MAR	TomTom	2011.06
MOZAMBIQUE	MOZ	Pitney Bowes	C. 2006
MYANMAR	MMR	Pitney Bowes	C. 2006
NEPAL	NPL	Pitney Bowes	C. 2006
NETHERLANDS	NLD	TomTom	2011.06
NEW CALEDONIA	NCL	Pitney Bowes	C. 2006
NEW ZEALAND	NZL	GeoNames	2011.07
NICARAGUA	NIC	Pitney Bowes	C. 2006
NIGER	NER	Pitney Bowes	C. 2006
NIGERIA	NGA	Pitney Bowes	C. 2006
NORFOLK ISLAND	NFK	Pitney Bowes	C. 2006
NORTHERN MARIANA ISLANDS	MNP	GeoNames	2011.07
NORWAY	NOR	TomTom	2011.06
OMAN	OMN	Pitney Bowes	C. 2006
PAKISTAN	PAK	GeoNames	2011.07
PALAU	PLW	Pitney Bowes	C. 2006
PAPUA NEW GUINEA	PNG	Pitney Bowes	C. 2006
PARAGUAY	PRY	Pitney Bowes	C. 2006
PHILIPPINES	PHL	GeoNames	2011.07
PITCAIRN	PCN	Pitney Bowes	C. 2006
POLAND	POL	TomTom	2011.06

Country Name	ISO 3166 Country Code	Data Source	Vintage
PORTUGAL	PRT	TomTom	2011.06
PUERTO RICO	PRI	GeoNames	2011.07
REUNION	REU	GeoNames	2011.07
ROMANIA	ROU	Pitney Bowes	C. 2006
RUSSIAN FEDERATION	RUS	TomTom	2011.06
SAINT HELENA, ASCENSION AND TRISTAN DA CUNHA	SHN	Pitney Bowes	C. 2006
SAINT PIERRE AND MIQUELON	SPM	GeoNames	2011.07
SAN MARINO	SMR	TomTom	2011.06
SAUDI ARABIA	SAU	Pitney Bowes	C. 2006
SENEGAL	SEN	Pitney Bowes	C. 2006
SINGAPORE	SGP	TomTom	2011.06
SLOVAKIA (SLOVAK REPUBLIC)	SVK	TomTom	2011.06
SLOVENIA	SVN	TomTom	2011.06
SOUTH AFRICA	ZAF	GeoNames	2011.07
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	SGS	Pitney Bowes	C. 2006
SPAIN	ESP	TomTom	2011.06
SRI LANKA	LKA	GeoNames	2011.07
SUDAN	SDN	Pitney Bowes	C. 2006
SWAZILAND	SWZ	Pitney Bowes	C. 2006
SWEDEN	SWE	GeoNames	2011.07

Country Name	ISO 3166 Country Code	Data Source	Vintage
SWITZERLAND	CHE	TomTom	2011.06
TAIWAN, PROVINCE OF CHINA	TWN	TomTom	2011.06
TAJIKISTAN	TJK	Pitney Bowes	C. 2006
THAILAND	THA	TomTom	2011.06
TIMOR-LESTE	TLS	Pitney Bowes	C. 2006
TUNISIA	TUN	Pitney Bowes	C. 2006
TURKEY	TUR	TomTom	2011.06
TURKMENISTAN	TKM	Pitney Bowes	C. 2006
TURKS AND CAICOS ISLANDS	TCA	Pitney Bowes	C. 2006
UKRAINE	UKR	Pitney Bowes	C. 2006
UNITED ARAB EMIRATES	ARE	Pitney Bowes	C. 2006
UNITED KINGDOM	GBR	TomTom	2011.06
UNITED STATES	USA	TomTom	2011.06
URUGUAY	URY	Pitney Bowes	C. 2006
UZBEKISTAN	UZB	Pitney Bowes	C. 2006
VATICAN CITY STATE (HOLY SEE)	VAT	TomTom	2011.06
VENEZUELA	VEN	Pitney Bowes	C. 2006
VIET NAM	VNM	Pitney Bowes	C. 2006
VIRGIN ISLANDS (U.S.)	VIR	GeoNames	2011.07
WALLIS AND FUTUNA ISLANDS	WLF	Pitney Bowes	C. 2006
WESTERN SAHARA	ESH	Pitney Bowes	C. 2006

Country Name	ISO 3166 Country Code	Data Source	Vintage
ZAMBIA	ZMB	Pitney Bowes	C. 2006

Spectrum Geocoding Scenarios

You can use Enterprise Manager to create dataflows that are appropriate for your business requirements and for the nature and quality of your data.

Multiple Country Stage with Geocode Address World Component

Let's assume the following scenario:

- Your input file includes addresses for six countries: Argentina (ARG), Brazil (BRA), Mexico (MEX), Chile (CHL), Venezuela (VEN), and Panama (PAN).
- Geocoders for three of these countries (ARG, BRA, and MEX) are deployed in a multiple country stage .
- Geocode Address World is deployed in a separate stage to geocode addresses that could not be identified by the country-specific geocoders.
- Your stage uses conditional routers (and optionally stream combiner) to manage the geocoding flow.

Step 1: Read input into the multiple country stage. Geocoded addresses can be written out to a file or optionally sent to the stream combiner.

Step 2: Some Addresses that could not be geocoded in step 1. This may be because they were addresses from CHL, VEN, or PAN, and these countries that were not included in the first stage. Or they may have failed to return a close match candidate in the first stage because of input errors or ambiguities in the address. These ungeocoded addresses are sent to the Geocode Address World stage.

Step 3 Addresses can be geocoded to postal or geographic accuracy by Geocode Address World. Successfully geocoded addresses can be written out to a file or optionally sent to the stream combiner.

Postal geocoded candidates will have a Z1 result code. Postal geocoded results may be very accurate in countries with robust postcode systems. See **Postal Geocoding** on page 426. Geographic candidates will have a G result code (for example G3 for a town/city match). See **Geographic Geocoding** on page 427.

Step 4: The stream combiner (if used in your dataflow) can combine all geocoded addresses and write them to a file or direct them for further processing.

This is one scenario. You could use Enterprise Manager to design more complex dataflows that are suitable for your needs.

Input

GeocodeAddressWorld takes an address as input. To obtain the best performance and the most possible matches, your input address lists should be as complete as possible, free of misspellings and incomplete

addresses, and as close to postal authority standards as possible. Most postal authorities have websites that contain information about address standards for their particular country.

Note: The country name or two- or three- character country ISO code is optional. If you omit the country, GeocodeAddressWorld returns the best available candidates based on the other input provided.

Input Fields

The following table provides information on the format and layout of GeocodeAddressWorld input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 114: GeocodeAddressWorld Input Data

columnName	Format	Description / Valid Values
AddressLine1	String	The first address line. For example, 4360 DUKES RD: **4360 DUKES RD** KALGOORLIE WA 6430
AddressLine2	String	The second address line of a two-line address. For example, Level 6 51 Jacobson St: 26 WELLINGTON ST E **SUITE 500** TORONTO ON M5E 1S2 This field is not used in Australia, Austria, Belgium, Brazil, Denmark, Finland, France, Germany, Ireland, Italy, Liechtenstein, Luxembourg, Malaysia, The Netherlands, Poland, Portugal, Spain, Sweden, Switzerland, and Thailand.
City	String	The city or town name. Your input address should use the official city name. This will produce the best geocoding results. For Thailand, this field contains the subdistrict (tambon).
County	String	The name of one of the following depending on the country: • **Not used**—AUT, BRA, CAN, FIN, GBR, MYS, PRT, SGP. • **Department**—FRA • **District (amphoe)**—THA • **District (fylke/counties)**—NOR • **District (poviat)**—POL • **Kommun**—SWE • **Kreis**—DEU • **Local Government Authority (LGA)**—AUS • **Province**—BEL, CHE, DNK, ESP, IRL, ITA, LIE, LUX, NLD • **Region**—NZL

columnName	Format	Description / Valid Values
FirmName	String	Company or name or place name. For example, PITNEY BOWES. **PITNEY BOWES** 4360 DUKES RD KALGOORLIE WA 6430
LastLine	String	The last line of the address. For example, KALGOORLIE WA 6430: 4360 DUKES RD **KALGOORLIE WA 6430**
Locality	String	The name of one of the following depending on the country: • **Not used**—AUS, AUT, BEL, CHE, DEU, DNK, FIN, FRA, IRL, LIE, LUX, MYS, NLD, NOR, POL, SGP, SWE, THA • **Dissemination Area and Enumeration Area (DA and EA)**—CAN • **Locality**—BRA, GBR, ITA, PRT • **Suburb**—NZL
PostalCode	String	The postal code in the appropriate format for the country.
StateProvince	String	The name of one of the following depending on the country: • **Not used**—BEL, CHE, DNK, IRL, LIE, LUX, NLD, NOR, SGP • **Bundesland**—DEU • **Province**—CAN • **Province (changwat)**—THA • **Province (voivodship)**—POL • **Region**—AUT, ESP, FRA, GBR, NZL, PRT • **Region (län)**—FIN • **Region (lan)**—SWE • **State**—AUS, BRA • **State (negeri)**—MYS
Country	String	The two- or three-character ISO country code. This field is optional. If you omit the country, GeocodeAddressWorld returns the best available candidates based on the other input provided For a list of the ISO codes see **Country ISO Codes and Module Support** on page 908.

Address Aliases

Some countries have alternative administrative names. For example, there may be an official name for a city or town, but there may also be common but unofficial alternative name for the same city or town. If alias information is available in the source data, Geocode Address World includes this alias in the database. This enables Geocode Address World to geocode successfully when alternative names are used in input addresses.

Language Aliases on page 454 are also supported.

Language Aliases

Some countries have more than one official or prominent language. For example, the same town may be commonly known by both German and Italian names. If language alias information is available in the source data, Geocode Address World uses this in the database. This enables Geocode Address World to geocode successfully when alternative language names are used in input addresses.

Aliases can exist for all administrative levels, from StateProvince state/province to Locality locality. See **Administrative Divisions and Postal Codes** on page 455 for a description of administrative levels associated with the geographic data.

Address Aliases on page 454 are also supported for commonly used, alternative administrative areas.

State or Province Abbreviations

In some countries, the state or province is an important part of the address and often this address component is abbreviated. For selected countries, these state/province abbreviations are recognized by Geocode Address World. For example, in the United States each state has a two-letter abbreviation (such as CA for California). Similarly, Netherlands, state abbreviations (such as GLD for Gelderland) are recognized.

Geocode Address World accepts state/province abbreviations for the following countries:

Table 115: Country State/Province Abbreviation Support

Country Name	State Or Province Division	Example
Australia (AUS)	StateProvince (State)	NSW (abbreviation for New South Wales
Canada (CAN)	StateProvince (Province)	AB (abbreviation for Alberta)
Italy (ITA)	County (Province)	MO (abbreviation for Modena)
Mexico (MEX)	StateProvince (State)	JA (abbreviation for Jalisco)
Netherlands (NLD	County (State)	FR (abbreviation for Friesland)
United States (USA)	StateProvince (State)	CA (abbreviation for California)

Geocode Address World evaluates these state or province abbreviations to better identify close matches. See **Geographic Geocoding with State/Province Abbreviation** on page 428 for an example that illustrates this feature.

Administrative Divisions and Postal Codes

Typical input addresses consist of street address, administrative division, and postal code information. Geocode Address World uses the administrative divisions and postal codes for geographic or postal geocoding.

- StateProvince (state or province)
- County (county, region, or district)
- City (town or city)
- Locality (locality suburb, or village)
- postal code

Specific administrative divisions vary by country. For example, Locality may contain locality, suburb, or barrio, depending on the country. StateProvince may contain state, province, region, or some other name depending on the country. See **State or Province Abbreviations** on page 454 for more information on how state/province abbreviations are interpreted by Geocode Address World.

Not all administrative divisions are used in addressing conventions for all countries. For example, in the USA, County (county) is not typically used in addresses. But for some countries, County is an important part of the address.

If your input data includes postal codes, Geocode Address Worldcan use this for postal geocoding, assuming that the source data includes postal data for the specific country.

Input Recommendations

You can optimize Geocode Address World results if you prepare and understand your input records. Follow these guidelines :

- Ensure that your input addresses are as complete and accurate as possible. If there are errors in your input addresses, Geocode Address World may still be able to geocode those addresses, but there may be more than one possible match or you may get non-close matches. If you can verify and correct any incomplete or inaccurate input addresses, you can get better results.
- Include postcodes in your input addresses if you have them. This is not required, but it allows Geocode Address World to perform postal geocoding. This may give you more accurate results for some addresses, depending on the country and on the completeness and accuracy of other address components.
- Include the country name or official three-character or two-character country ISO code in your input addresses. This is not required, but it may help Geocode Address World distinguish between similar addresses and city names that may occur in different countries.
- Format your input addresses consistently. Geocode Address World can handle input addresses in a wide variety of input formats, or can handle unformatted (single line) input. But you can get more accurate and faster results if your input addresses are consistently formatted and conform to country-specific address conventions. Even if your input address are single line (unformatted), you may get better results and performance if the address components are ordered consistently. Use the AddressLine1 input area for unformatted addresses. See **Single Line Input** on page 455

Single Line Input

Address input can be formatted into separate input fields or input can as single line input. Use AddressLine1 to enter single line input.

Single Line Geographic Geocoding

In this example, unformatted (single line) input is used. Geocode Address World analyzes single line input to identify the geographic address components (Graz in this example), and then geocodes to a geographic centroid. The MainAddress (street information) is not used.

Sackstraße 10 Graz

Geocode Address World returns a geographic close match candidate based on an City match. Even though the country was not specified, Geocode Address World identifies the close match in Austria (AUT).

StateProvince: Steirmark
County: Graz (Stadt)
City: Graz
Country: AUT
Result Code: G3
X: 15.44172
Y: 47.06792

If your input addresses are accurate, unformatted input can produce a match rate comparable to that of formatted input. However, geocoding unformatted addresses typically has slower performance than geocoding formatted addresses.

Single Line Postal Geocoding with Country Specified

In this example, single line input is used and a postcode is provided. The country Austria (AUT) is also specified. The street address is also input, but this is ignored by Geocode Address World.

Alpenstraße 117 5020 AUT

Geocode Address World returns a postal centroid close match candidate (Z1 result code). Because the country (AUT) is specified in the input, the country must be matched and a single close match for that postal code in Austria is returned. Non-close matches with the 5020 postal code from other countries are also returned.

StateProvince: Salzburg
Country: AUT
Postcode: 5020
Result Code: Z1
X: 13.04685
Y: 47.80262

Options

GeocodeAddressWorld allows you to set default processing options through the Management Console. You can override certain settings for individual calls to GeocodeAddressWorld using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

Geocoding Options

The following table lists the options that control how a location's coordinates are determined.

Table 116: Geocoding Options

optionName	Description / Valid Values
CoordinateSystem	A coordinate system is a reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry. Spectrum™ Technology Platform supports systems recognized by the European Petroleum Survey Group (EPSG). One the following: **EPSG:4283** Also known as the GDA94 coordinate system. **EPSG:4326** Also known as the WGS84 coordinate system. Default.

Matching Options

Table 117: Matching Options

optionName	Description / Valid Values
KeepMultimatch	Specifies whether to return results when the address matches to multiple candidates in the database. If this option is not selected, an address that results in multiple candidates will fail to geocode. If you select this option, specify the maximum number of candidates to return using the MaxCandidates option (see below). One of the following: **Y** Yes, return candidates when multiple candidates are found. Default. **N** No, do not return candidates. Addresses that result in multiple candidates will fail to geocode.
MaxCandidates	If you specify KeepMultimatch=Y, this option specifies the maximum number of results to return. The default is 1.
CloseMatchesOnly	Specifies whether to return only those geocoded results that are close match candidates. For example, if there are 10 candidates and two of

optionName	Description / Valid Values
	them are close candidates, and you enable this option, only the two close matching candidates would be returned instead of all 10. One of the following: **Y** Yes, return only close matches. **N** No, do not return only close matches. Default.

Data Options

The Data tab allows you to specify which databases to use in geocoding. Databases contain the address and geocode data necessary to determine the geocode for a given address. The data is based on address and geocoding data from postal authorities and suppliers of geographical data.

Table 118: Data Options on page 458 lists the options available for specifying which databases to use and the search order of databases.

Table 118: Data Options

optionName	Description / Valid Values
DatabaseSearchOrder	The name of one or more database resources to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. You can specify multiple database resources. If you specify more than one database, list them in order of preference. The order of the database has an effect when there are close match candidates from different databases. The close matches that are returned come from the database that is first in the search list. Close matches from lower ranked databases are demoted to non-close matches.

Output

GeocodeAddressWorld returns the latitude/longitude, city, county, and result indicators. Result indicators describe how well the geocoder matched the input to a known location and assigned a latitude/longitude; they also describe the overall status of a match attempt. The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Address Output

Table 119: Address Output

columnName	Description
City	Municipality name.
Country	The three-letter ISO 3166-1 Alpha 3 country code. The two-letter code can also be used. See **Country Geographic Data Coverage** on page 430for a list of countries and data sources for geographic geocoding. See **Country Postal Data Coverage** on page 443for a list of postal geocoding countries and data sources.
County	This field contains an area that is smaller than a state/province but larger than a city. The specific area varies by country: • AUS—Local Government Authority (LGA) • AUT—Province • BEL—Province • BRA—Not used • CAN—Not used • CHE—Province • DEU—Kreis • DNK—Province • FIN—Province (kommune) • FRA—Department • GBR—County • ITA—Province • LIE—Province • LUX—Province • MYS—District (daerah) • NLD—Province • NZL—Not used • POL—District (poviat) • PRT—Not used • SGP—District • SWE—Region (kommun) • THA—District (amphoe)
PostalCode	The postcode for the address. The format of the postcode varies by country.

columnName	Description
StateProvince	The meaning of StateProvince varies by country: • AUS—State • AUT—Region • BEL—Not used • BRA—State • CAN—Province • CHE—State • DEU—Bundesland • DNK—Not used • ESP—Region • FIN—Region (län) • FRA—Region • GBR—Region • IRL—Not used • ITA—Region • LIE—State • LUX—Not used • MYS—State (negeri) • NLD—Not used • NOR—Not used • NZL—Region • POL—Province (voivodship) • PRT—Region • SGP—Not used • SWE—Region (lan) • THA—Province (changwat)

Geocode Output

Table 120: Geocode Output

columnName	Description
CoordinateSystem	The coordinate system used to determine the latitude and longitude coordinates. A coordinate system specifies a map projection, coordinate units, etc. An example is EPSG:4326. EPSG stands for European Petroleum Survey Group.

columnName	Description
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

Result Codes

Result codes contain information about the success or failure of the geocoding attempt, as well as information about the accuracy of the geocode.

Table 121: Result Code Output

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. For more information, see **Introduction** on page 597.
IsCloseMatch	Indicates whether or not the address is considered a close match. An address is considered close based on the "Close match criteria" options on the Matching tab. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
MultiMatchCount	For street address geocoding, the number of matching address positions found for the specified address. For intersection geocoding, the number of matching street intersection positions found for the specified addresses.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	If the geocoder could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data

columnName	Description
	• Multiple Matches Found • Exception occurred • Unable to initialize Geocoder • No Match Found
Status.Description	If the geocoder could not process the address, this field will show a description of the failure. **Problem + explanation** — Returned when Status.Code = Internal System Error. **Geocoding Failed** — Returned when Status.code = No Geocode Found. **No location returned** — Returned when Status.code = No Geocode Found. **No Candidates Returned** — The geocoder could not identify any candidate matches for the address. **Multiple Candidates Returned and Keep Multiple Matches not selected** — The address resulted in multiple candidates. In order for the candidate address to be returned, you must specify `KeepMultimatch=Y`.
LocationPrecision	A code describing the precision of the geocode. One of the following: **0** — No coordinate information is available for this candidate address. **1** — Interpolated street address. **2** — Street segment midpoint. **3** — Postal code 1 centroid. **4** — Partial postal code 2 centroid. **5** — Postal code 2 centroid. **6** — Intersection. **7** — Point of interest. **8** — State/province centroid. **9** — County centroid. **10** — City centroid. **11** — Locality centroid.

columnName	Description	
	12	Additional point precision for unspecified custom item.
	13	Additional point precision for unspecified custom item.
	14	Additional point precision for unspecified custom item.
	15	Additional point precision for unspecified custom item.
	16	The result is an Address Point.
	17	The result was generated by using address point data to modify the candidates segment data.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.	

Geographic Candidate Ranking

Identical geographic area names can be found in many countries. When this occurs, Geocode Address World uses a ranking system to determine which of potential candidates is the most likely close match.

The specific details of this weighted ranking depend somewhat on the data source (TomTom, GeoNames, or Pitney Bowes source), but the following criteria are weighted to determine the most likely close match candidate.

- country capital
- administrative area (state/province, regional, county) capital
- population range

Country capital status outweighs any other geographic ranking criteria. For example, San Juan entered as City returns San Juan, Puerto Rico (PRI) as the close match because it is the capital of PRI. Other San Juan cities in the world (including Spain, Costa Rica, Dominican Republic and Philippines) can be returned as non-close matches regardless of their population. To return matches, you must check the Keep multiple matches check box in Matching Options of the Management Console, and specify the number of matches you want to return.

Similarly, Geocode Address World returns Roma, ITA as a close match since that is the capital of Italy, but Roma in Romania, Honduras, and Panama are returned as non-close matches.

State/province administrative capitals are highly weighted even if their populations are not very large. For example, Springfield returns a close geographic match to Springfield, Illinois USA because this is the state capital of Illinois. Springfield, Massachusetts has a somewhat larger population, but this is outweighed by the state capital status of Springfield Illinois. Other less populous Springfield communities in the USA and other countries are also returned as close matches, but are listed below the Springfield Illinois candidate. It is possible for a large city to rate as an equal close match along with a smaller, identically named state/province capital. However, the state/province capital will not be demoted, even if it has a relatively small population.

Similarly, if your input is Albany in City with no country specified, Geocode Address World returns Albany, NY, USA as the close match candidate. This is because Albany is the capital of New York State, and therefore gets a high ranking as administrative area capital. The population is also a contributing ranking factor. If you specify the city of Albany with a different country, such as New Zealand, then the country is used and Albany, NZL is returned as the close match candidate.

Match Codes

Matches in the G category indicate that the candidate is located at the geographic centroid with the following possible accuracy levels. Not all levels of accuracy are possible for all countries.

- **G0**—Country centroid. This is not returned for GeocodeAddressWorld.
- **G1**—State or province centroid. For Japan, this indicates a prefecture (ken) match.
- **G2**—County centroid. For Japan, this indicates a city (shi) match.
- **G3**—City centroid. For Japan, this indicates a municipality subdivisio (oaza) match. For Australia, Local Government Authority (LGA) information can be returned from the Street Range Address Database only (not the G-NAF database).
- **G4**—Locality centroid. For Japan, this indicates a city district (chome) match.

Matches in the Z category indicate that no street match was made for one of the following reasons:

- You specified to match to postal code centroids. The resulting point is located at the postal code centroid with four possible accuracy levels.
- There is no close match and you specified to fall back to postal code centroid

The Z category contains the following accuracy levels:

- **Z0**—Postal Code match, no coordinates available (rare occurrence).
- **Z1**—Postal Code centroid match.
- **Z3**—Full postal code centroid match. For Canada, this is an FSALDU centroid.
- **Z6**—Postal Code centroid match for point ZIP.

GeocodeUSAddress

GeocodeUSAddress takes an address and returns latitude/longitude coordinates. GeocodeUSAddress also standardizes and validates addresses using data from the U.S. Postal Service.

GeocodeUSAddress can also geocode intersections. Instead of entering a mailing address, you can enter and intersection such as "Pearl St. and 28th" and obtain the coordinates of the intersection.

GeocodeUSAddress is part of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Input

GeocodeUSAddress takes an address as input. To obtain the best performance with GeocodeUSAddress and the most possible matches, your input address should be as complete as possible and free of misspellings and incomplete information. Input addresses should be as close to USPS standards as

possible for the highest match rate. For information on USPS standards, see the USPS website **http://www.usps.com** .

Input addresses should contain a street address line and a lastline, or a single line with both address and lastline elements. This helps GeocodeUSAddress accurately identify an area in which to search for a match candidate, based on the city, state, and ZIP Code.

GeocodeUSAddress also accepts a street address line with individual city, state, and ZIP Code lines instead of a last line. You should only use this type of input if you are confident that the input address is free of misspellings and incomplete information.

If you are using GeocodeUSAddress for address standardization, input addresses must have at least a street name, and either a city and state or a ZIP Code to obtain a match. If you are using GeocodeUSAddress to obtain geocoding information, input addresses only need to contain a ZIP + 4 Code to receive geocoding information.

The following table provides information on the format and layout of GeocodeUSAddress input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 122: GeocodeUSAddress Input Data

columnName	Format	Description / Valid Values
AddressLine1	String	The first address line or a street intersection. To specify a street intersection, use and, &, at, or @. For example, PEARL & 28th. GeocodeUSAddress does not match intersections when processing in CASS mode. You may enter an address range instead of an individual address number. For example, 10-12 FRONT ST. For additional information, see **Address Range Matching** on page 468.
AddressLine2	String	The second address line or a street intersection. To specify a street intersection, use and, &, at, or @. For example, PEARL & 28th.L GeocodeUSAddress does not match intersections when processing in CASS mode.
AddressLine3	String	Third address line.
AddressLine4	String	Fourth address line.
AddressLine5	String	Fifth address line.
AddressLine6	String	Sixth address line.

columnName	Format	Description / Valid Values
City	String	The name of the municipality, such as a city or town. **Note:** If there is any data in the input fields AddressLine3, AddressLine4, AddressLine5, or AddressLine6, GeocodeUSAddress will ignore data in the City input field.
FirmName	String	The name of a business. GeocodeUSAddress attempts to match the input firm name to the recognized firm names in the USPS data for a higher quality match. If the firm name is not in the USPS data, GeocodeUSAddress ignores the firm name when matching and returns the firm name with the output.
LastLine	String	The last line of an address containing the city, state, and ZIP Code.
PostalCode	String	The 5-digit ZIP Code or the 9-digit ZIP + 4 code. **Note:** If there is any data in the input fields AddressLine3, AddressLine4, AddressLine5, or AddressLine6, GeocodeUSAddress will ignore data in the PostalCode input field.
StateProvince	String	The name or abbreviation of the state. **Note:** If there is any data in the input fields AddressLine3, AddressLine4, AddressLine5, or AddressLine6, GeocodeUSAddress will ignore data in the StateProvince input field.

How GeocodeUSAddress Processes Addresses

GeocodeUSAddress processes addresses in the following order:

1. Parses the address elements.

   GeocodeUSAddress parses input address data into single elements. Parsing occurs on data in the order in which you load the data. Even if a valid address is missing an element, GeocodeUSAddress can find a match. Some elements, such as predirectionals, may not be critical elements of some addresses. By comparing an address as input against all known addresses in a search area, GeocodeUSAddress can usually determine if any of these elements are missing or incorrect.

2. Finds possible matches within the search area.

GeocodeUSAddress uses the last line elements of an address to determine a search area. You can specify if you want the search area based on a finance area or on an area defined by the city, state, and ZIP Code. (A Finance Area is a collection of ZIP Codes within a contiguous geographic region.) If the city and state are not in the ZIP Code, GeocodeUSAddress performs separate searches for the ZIP Code and city.

After GeocodeUSAddress has determined the search area, it tries to match the elements from the street address line to the records in the standardized data files and does the following:

- Checks input address ranges for missing or misplaced hyphens, and alpha-numeric ranges for proper sequence.
- Searches for any misspellings and standard abbreviations. For example, the GeocodeUSAddress can recognize Mane for Main and KC for Kansas City.
- Searches for any alias matches to the USPS and Spatial data (TIGER and GDT). For example, GeocodeUSAddress recognizes that in Boulder, CO Highway 36 is know as 28th Street.
- Searches for any USPS recognized firm names for additional match verification.
- Searches for street intersection matches. Matching to an intersection is extremely useful when you are using address matching to obtain a geocode.
- Searches for addresses lines that contain a house number and unit number as the same element. For example, GeocodeUSAddress recognizes the input 4750-200 Walnut Street and performs recombination to output 4750 WALNUT ST STE 200.

Note: The USPS does not consider intersections valid addresses for postal delivery. Therefore, the GeocodeUSAddress does not match intersections when processing in CASS mode.

3. Scores each possible match against the parsed input.

   GeocodeUSAddress compares each element in the input address to the corresponding element in the match candidates, and assigns a confidence level. GeocodeUSAddress weighs the confidence level for all of the elements within a match candidate, and assigns a final score to the sum.

   Note: GeocodeUSAddress uses a penalty scoring system. If an element does not exactly match an element in the match candidate, the GeocodeUSAddress adds a penalty to the score of the match candidate. Therefore, scores with lower numbers are better matches.

4. Determines the match.

   GeocodeUSAddress prioritizes each match candidate based on the assigned confidence score and returns as a match the candidate that has the lowest score.

   The match mode you choose determines the range that GeocodeUSAddress allows for a match. GeocodeUSAddress only returns a match if the score of the target address falls within the range designated by the selected match mode.

   In some cases, more than one match candidate may have the lowest score. In this instance, GeocodeUSAddress cannot determine on its own which record is correct, and returns a status indicating multiple matches.

   Note: If you have enabled Delivery Point Validation (DPV) processing, GeocodeUSAddress automatically attempts to resolve multiple matches using DPV.

   Along with a standardized address, GeocodeUSAddress also returns the following:

   - Geocode—Longitude and latitude for the address
   - Match code—Information about the match of the input address to the reference data
   - Location code—Precision level of a geocode
   - Parity—The side of the street on which the match resides.

GeocodeUSAddress does not return parity when processing in relaxed mode. For more information on GeocodeUSAddress output, see **Output** on page 483.

Address Range Matching

Some business locations are identified by address ranges. For example, a shopping plaza could be addressed as 10-12 Front St. This is how business mail is typically addressed to such a business location. These address ranges can be geocoded to the interpolated mid-point of the range.

Address ranges are different from hyphenated (dashed) addresses that occur in some metropolitan areas. For example, a hyphenated address in Queens County (New York City) could be 243-20 147 Ave. This represents a single residence (rather than an address range) and is geocoded as a single address. If a hyphenated address returns as an exact match, GeocodeUSAddress does not attempt to obtain an address range match.

Address range matching is not available in Exact or CASS modes, since an address range is not an actual, mailable USPS® address. The following fields are not returned by address range geocoding:

- ZIP + 4® (in multiple segment cases)
- Delivery point
- Check digit
- Carrier route
- Record type
- Multi-unit
- Default flag

Address range matching works within the following guidelines:

- There must be two numbers separated by a hyphen.
- The first number must be lower than the second number.
- Both numbers must be of the same parity (odd or even) unless the address range itself has mixed odd and even addresses.
- Numbers can be on the same street segment or can be on two different segments. The segments do not have to be contiguous.
- If both numbers are on the same street segment, the geocoded point is interpolated to the approximate mid-point of the range.
- If the numbers are on two different segments, the geocoded point is based on the last valid house number of the first segment. The ZIP Code and FIPS Code are based on the first segment.
- In all cases, odd/even parity is evaluated to place the point on the correct side of the street.

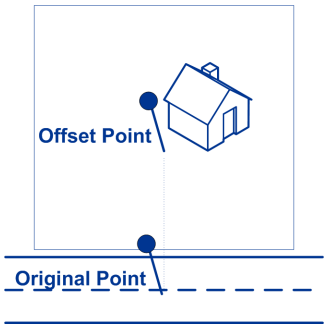
Options

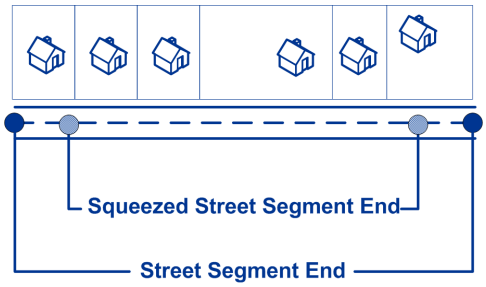
GeocodeUSAddress allows you to set default processing options through the Management Console. You can override certain settings for individual calls to GeocodeUSAddress using the Client API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

If you do not specify system-level defaults through the Management Console or a custom interface, and you do not pass any processing options in the job, Spectrum™ Technology Platform uses the default options.

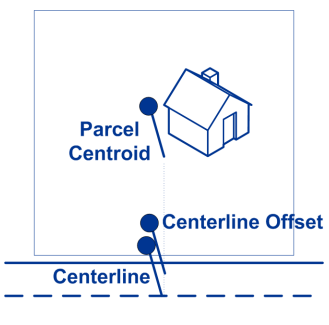
Geocoding Options

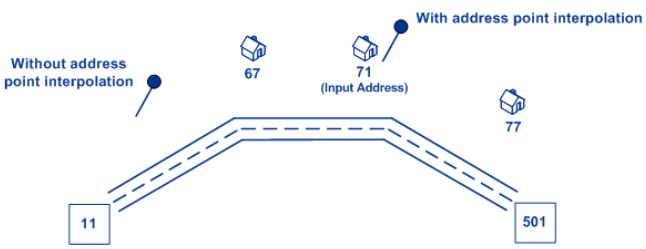
Table 123: GeocodeUSAddress Geocoding Options

optionName	Description / Valid Values
Dataset	The name of the database resource that contains the data to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.
Offset	Specifies the offset distance from the street segments, in feet. The range is 0 to 5280. The default value is 50. The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point.  Street coordinates are accurate to 10,000ths of a degree and interpolated points are accurate to the millionths of a degree.
Squeeze	Specifies the distance, in feet, to move the street segment end points toward the middle of the segment. Squeeze is used in street-level matching. Use the squeeze setting to prevent address points from residing in an intersection or too close to the end of a street. The range is 0 to 2147483647. The default value is 50.

optionName	Description / Valid Values
	The following diagram compares the end points of a street to squeezed end points.  Squeezing the street segment endpoints affects street-level matching by reducing the length of a street segment, thereby reducing the spacing between address points along the segment. For example, if the length of a street segment is 1,000 feet and there are 10 addresses along the segent, street-level matching would result in each address being spaced 100 feet apart (1,000 ÷ 10). If you were to set a squeeze value of 100 feet, moving each street segment endpoint 100 feet torward the center of the street segment, the length of the street segment would be reduced to 800 feet (reduced by 100 feet on each end). Street-level matching would then result in addresses beging spaced 80 feet apart (800 ÷ 10).
LatLonFormat	Specifies the format of the latitude/longitude returned by GeocodeUSAddress. **Decimal** — The latitude/longitude is returned in decimal format (default). For example: 90.000000-180.000000 **Integer** — The latitude/longitude is returned in integer format. For example: 90000000-180000000
CentroidPreference	Determines the type of centroids returned by GeocodeUSAddress. A centroid is the center of an area. The centroid coordinates are the average of the sets of coordinates that describe the area. **NoCentroids** — Do not return centroids. If an address-level geocode cannot be determined, do not attempt to determine a centroid. **AllCentroids** — Return ZIP Code centroids only. If you select this option, address-level geocodes will not be returned. **AddressUnavailable** — Return a ZIP Code centroid if an address-level geocode cannot be determined (default).

optionName	Description / Valid Values
FallbackToStreet	Specifies whether to attempt to return a street centroid when an address-level geocode cannot be determined. To determine a street centroid, GeocodeUSAddress searches the input ZIP Code or city for the closest match. If GeocodeUSAddress is able to locate the street, it returns a geocode along the matched street segment. For example, if the input address is 5000 Walnut Street, Boulder 80301, and there is no 5000 Walnut Street, GeocodeUSAddress searches for the closest match to that address within the ZIP Code 80301. If there were no input ZIP Code, GeocodeUSAddress would search for the closest match to the input address within Boulder. If the input address is Walnut Street, Boulder 80301, since there is no house number, GeocodeUSAddress searches for the street within the input ZIP Code. Street centroid geocodes are indicated by value in the LocationCode output field that begins with "C". For more information, see **Street Centroid Location Codes** on page 585. **Note:** This option is not available if you set MatchMode to CASS. One of the following: **Y** Yes, attempt to determine the street centroid when an address-level geocode cannot be determined. **N** No, do not attempt to determine the street centroid when an address-level geocode cannot be determined.
FallbackToGeographic	Specifies whether to attempt to return a city, county, or state centroid when an address-level geocode cannot be determined. The geocoder returns the most precise geographic centroid that it can based on the input. For example, if the input contains a valid city and state, a city centroid would be returned. **Note:** There are approximately 300 major cities that can be geocoded to a city centroid level even if a valid state is not provided in the input. Geographic centroid geocodes are indicated by value in the LocationCode output field that begins with "G". For more information, see **Geographic Centroid Location Codes** on page 591. **Note:** This option is not available if you set MatchMode to CASS. One of the following: **Y** Yes, attempt to determine the geographic centroid when an address-level geocode cannot be determined.

optionName	Description / Valid Values
	N No, do not attempt to determine the geographic centroid when an address-level geocode cannot be determined.
Datum	Determines the North American Datum to use when geocoding datum on the input value. Datum is the mathematical model of the Earth used to calculate the coordinates on any map, chart, or survey system. **NAD27** This datum does not include the Alaskan Islands or Hawaii. Latitudes and longitudes that are surveyed in the NAD27 system are valid only in reference to NAD27 and are not valid for maps outside the U.S. **NAD83** This datum is earth-centered and defined with satellite and terrestrial data. NAD83 is compatible with the World Geodetic System 1984 (WGS84), which is the terrestrial reference frame associated with the NAVSTAR Global Positioning System (GPS) used extensively for navigation and surveying. This is the default setting.
CenterlineOffset	The offset distance, in feet, used to calculate the street centerline coordinates. The default is 0. If you specify a value other than 0, GeocodeUSAddress calculates the street centerline coordinates by offsetting the centerline point by the distance you specify in the direction of the parcel centroid.  In an interpolated match, the centerline offset cannot be greater than the distance from the centerline to the interpolated address point. If you specify a centerline offset distance that is greater than this distance, the offset will be limited to the distance to the interpolated point. In effect, the centerline coordinates would be the same as the coordinates for the interpolated point.

optionName	Description / Valid Values
RetrieveElevation	Specifies whether GeocodeUSAddress returns the elevation of the address. Elevation is the distance above or below sea level of a given location. The elevation is returned in the **Elevation** output field, which is part of the Latitude/Longitude output group. **Note:** This option requires that you have licensed and installed the Centrus Premium Points database. Elevation data is not available for all addresses. See the coverage map included with the points database.
AddressPointInterpolation	Specifies whether to perform address point interpolation when an exact match for the address cannot be found in the geocoding database. Address point interpolation is a patented process that results in a more accurate interpolated point. It improves upon regular street segment interpolation by using point data in the interpolation process, as opposed to using street segments alone. **Note:** Address point interpolation is only available when using a point-level geocoding database. It is not available when using point addresses in an auxiliary file. One of the following: **Y** Yes, perform address point interpolation. **N** No, do not perform address point interpolation. The following illustration shows how address point interpolation works. In the example, the input house number is 71. The geocoding database contains address points for 67 and 77. The street segment has a range of 11 to 501. With address point interpolation, GeocodeUSAddress performs the interpolation for the input house number 71 using the points of 67 and 77. Without address point interpolation, GeocodeUSAddress performs the interpolation with the street segment end points of 11 and 501, resulting in a far less accurate result. 

Matching Options

Table 124: GeocodeUSAddress Matching Options

optionName	Description / Valid Values
AddressPreference	Determines which address to use when more than one address is present in the address block. **PreferBottom** — Uses the second line entered (default). You must select this value if you specify MatchMode=CASS. **PreferPOBox** — Uses the P.O Box. **PreferStreetAddress** — Uses the street address.
FirmNameSearch	Specifies whether GeocodeUSAddress should use firm name matching logic to enhance address matching. Firm matching logic matches a business name in the input to recognized business names. The input firm name does not need to be spelled correctly to obtain a match. GeocodeUSAddress uses a soundex algorithm to match the firm name. A suite or unit number is not required to make the match. **Note:** This type of match is not available when processing in CASS mode. One of the following: **Never** — Do not use firm matching (default). Note that GeocodeUSAddress may correct the firm name even if you specify **Never** if it can find a match using the address line data. **OnAddressLineFail** — Use firm matching only if a match cannot be determined using address matching. **Always** — Always attempt to match using firm name matching. If firm name matching fails, attempt to match using address matching.
FirstLetterSearch	Specifies whether to look for the correct first letter of a street name if the first letter is missing or incorrect. If enabled, GeocodeUSAddress searches through the alphabet looking for the correct first letter to complete the street address. **Note:** This option is not available if the match mode is set to exact. One of the following:

optionName	Description / Valid Values
	Y Perform first letter search. **N** Do not perform first letter search (default). This example includes an incorrect first letter: Input: 4750 nalnut boulder co 80301 Output: 4750 Walnut St Boulder CO 80301-2532 This example excludes a first letter: Input: 4750 alnut boulder co 80301 Output: 4750 Walnut St Boulder CO 80301-2532 This example includes an extra first letter: Input: 4750 wwalnut boulder co 80301 Output: 4750 Walnut St Boulder CO 80301-2532
BuildingSearch	Specifies whether GeocodeUSAddress attempts to obtain a street address when the input address contains a building name with no suite or unit number. When this option is disabled, GeocodeUSAddress is able to match to building names only if there is a unit number in the input. For example, if the building search option were disabled and you entered this input: 5001 Chrysler Bldg New York, NY 10174 GeocodeUSAddress would successfully return the street address: 405 Lexington Ave RM 5001 New York, NY 10174-5002 With this option enabled, GeocodeUSAddress is also able to obtain a street address when only a building name with no unit number is provided. For example, if you enable this option and provide this address: Chrysler Bldg New York, NY 10174 You will get the street address: 405 Lexington Ave New York, NY 10174-00 **Note:** This type of match is not available when processing in CASS mode.

optionName	Description / Valid Values
	One of the following: **Y** Use firm name matching logic (default). **N** Do not use firm name matching logic.
RetrieveAPN	Specifies whether GeocodeUSAddress should determine the address's APN (assessor's parcel number). The APN is an ID number assigned to a property by the local property tax authority. The APN is returned in the **APN** output field, which is part of the Census output group. **Note:** This option requires that you have licensed and installed the Cenrus Enhanced Points or Centrus Premium Points database. APN data is not available for all addresses. See the coverage map included with the points database.
PerformDPV	Specifies whether GeocodeUSAddress should process addresses using Delivery Point Validation (DPV). DPV is a United States Postal Service (USPS) technology that validates the accuracy of address information down to the physical delivery point. You must have licensed the optional DPV processing option to use this feature. You must also install the DPV database. To use DPV, enable this processing option and specify D in OutputRecordType. One of the following: **Y** Perform DPV. **N** Do not perform DPV (default). If you use DPV, GeocodeUSAddress automatically resolves multiple matches. False-positive addresses, also known as seed records, are addresses the USPS monitors to ensure users are not attempting to create a mailing list from the DPV data. If GeocodeUSAddress matches an address in your input data to a false-positive address, you receive a message indicating you have encountered a false-positive address. Processing continues to the end of your job, but DPV processing is not available for this job and subsequent jobs until you have reported the false-positive address encounter to technical support and have received a new security key.
PerformLACSLink	Specifies whether GeocodeUSAddress should process addresses using LACS^{Link}. One of the following:

optionName	Description / Valid Values
	Y Perform LACS^{Link} **N** Do not perform LACS^{Link} (default). If you use LACS^{Link}, be sure to choose to specify output record types P and Q so that the fields USLACS, USLACS.ReturnCode, and LACSADDRESS are included in the output. For more information, see **Locatable Address Conversion System (LACS)** on page 303.
PreferZipCodeOverCity	Specifies whether to prefer candidates that match the input ZIP over candidates that match to input city. **Note:** This option is not available when processing in CASS mode. One of the following: **Y** Perfer candidates that match the input ZIP Code. **N** Prefer candidates that match the input city (default). For example, consider this input address: 301 BRYANT ST SAN FRANCISCO CA 94301 Without this option enabled, the best match would be the one that matches the input city name: 301 BRYANT ST SAN FRANCISCO CA 94107-4167 With this option enabled, the best match would be the one that matches the input ZIP Code: 301 BRYANT ST PALO ALTO CA 94301-1408
KeepMultimatch	Select this option to return the list of possible matches when Geocode US Address finds more than one possible match for the input address and cannot identify a single best match. One of the following: **Y** Return the addresses that are possible matches for the input address (default). **N** Do not return the ambiguous matches.

optionName	Description / Valid Values
KeepCandidates	Select this option to return candidate addresses whenever the match attempt produces candidates. If you enable this option, the geocoder will return candidates both when the input address matches to a single address and when the input address matches multiple addresses. This option differs from `KeepMultimatch` in that the `KeepMultimatch` option does not return candidates if the input address matches to a single address. One of the following: **Y** Return candidates for all match attempts. **N** Do not return candidates for all matches (default).
CloseMatchesOnly	If you specify `KeepCandidates=Y` you can choose to return just those candidates that are considered to be a close match. The criteria used to determine whether a candidate is a close match are those you specify in the MatchMode option. **Note:** A close match does not necessarily indicate a high-quality match. You should always check the values in the output fields MatchCode and LocationCode to determine the quality of the match. One of the following: **Y** Return close match candidates only (default). **N** Return all candidates.
MatchMode	Determines the leniency used to find a match. One of the following: **Custom** Allows you to select the specific criteria to use when matching the input address to an address in the postal database. **Exact** Requires a very tight match. This is a restrictive mode that generates the fewest number of match candidates to search, which decreases the time to obtain a match. When using this mode, ensure that your input address list is very clean; free of misspellings and incomplete addresses. **Close** Requires a moderately confident match. Generates a moderate number of match candidates. **Relax** Default. This is the loosest match mode and generates the most match candidates, which increases the processing time and results in more multiple matches. Use this mode if your address list may contain misspellings and incomplete

optionName	Description / Valid Values
	addresses. This is the only mode that does not respect the street parity for an address match. **CASS** Imposes additional rules to ensure compliance with the USPS regulations for CASS. The purpose of this mode is to create a list of mailable addresses. This mode generates a large number of match candidates. This mode deviates from the other modes in its processing. This mode does not perform intersection, building name, or spatial alias (TIGER and GDT street name alias) matches. It also does not match to candidates from data sources that do not have USPS equivalent records. This mode recognizes and parses two unit numbers on the same address line, for example a building and unit number.
MustMatchInput	Specifies whether candidates must match all non-blank input fields. For example, if an input address contains a city and postal code, then candidates for this address must match the city and postal code. One of the following: **Y** Yes, candidates must match all input. **N** No, candidates do not have to match all input. Default.
MustMatchHouseNumber	Specifies whether candidates must match the house number. If the input house number is not within a range from the street, GeocodeUSAddress selects the nearest range on the street which has the same parity (even or odd house number) as the input address number. GeocodeUSAddress returns one or more of the closest matches inside this range that preserves street parity. This requires GeocodeUSAddress to change the house number. The new house number is equal to one of the range's endpoints, possibly plus or minus one to preserve street parity. **Note:** Even when this option is disabled and an inexact match on the house number is found, GeocodeUSAddress still returns an error code. When this option is disabled and no exact matching house number is found, a match code of either E029 (no matching range, single street segment found), or E030 (no matching range, multiple street segment) is returned. GeocodeUSAddress does not change the house number on the output address. In order to access the inexact address number candidates, you must specify `KeepMultimatch=Y`. If there are inexact house

optionName	Description / Valid Values
	number candidates returned, the corresponding match codes begin with the letter 'H' indicating that the house number was not matched. Additionally, even when one or more exact candidates are found, inexact matches to the house number are still on the list of possible candidates, and these can be differentiated from the others by their Hxx match codes. For more information on match codes, see **Geocoding Match Codes** on page 592. One of the following: **Y** Yes, candidates must match the house number. **N** No, candidates do not have to match the house number.
MustMatchStreet	Specifies whether candidates must match the street name. One of the following: **Y** Yes, candidates must match the street name. **N** No, candidates do not have to match the street name.
MustMatchCity	Specifies whether candidates must match the city. If you do not require exact matches on city, the geocoder searches on the street address matched to the particular postal code, and considers other cities that do not match the name, but do match the postal code. One of the following: **Y** Yes, candidates must match the city. **N** No, candidates do not have to match the city.
MustMatchStateProvince	Specifies whether candidates must match the state. One of the following: **Y** Yes, candidates must match the state. **N** No, candidates do not have to match the state.
MustMatchPostalCode	Specifies whether candidates must match the postal code. If you do not require exact match on postal codes, the geocoder searches a wider area for a match. While this results in slower performance, the match rate is higher because the request does not need to match exactly when it compares match candidates. One of the following:

optionName	Description / Valid Values	
	Y	Yes, candidates must match the postal code.
	N	No, candidates do not have to match the postal code.

Difference Between Match Criteria for U.S. and Non-U.S. Geocoding

The "must match criteria" used in the custom match mode of Geocode US Address work differently than the "close match criteria" in non-U.S. geocoders. For Geocode US Address, the custom match criteria specify which address elements must match the reference database in order for the match to be returned as a candidate. All candidates returned by Geocode US Address will match the elements you specify as long as those elements are available in the reference database. However, in non-U.S. geocoders, the "close match" criteria are used to determine which candidates are close matches and which are non-close matches. Non U.S. geocoders can return both close candidates and non-close candidates, depending on whether you enable the `CloseMatchesOnly` option. In summary, the "must match" criteria used by Geocode US Address automatically limit the candidates returned, whereas the "close match criteria" used by non-U.S. geocoders do not limit the candidates returned.

Output Format

The following table lists the GeocodeUSAddress options that control the format of the output.

Table 125: GeocodeUSAddress Output Format Options

optionName	Description / Valid Values	
OutputCasing	Specifies the casing of the output data. One of the following:	
	M	The output in mixed case (default). For example: 123 Main St Mytown FL 12345
	U	The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputFormattedOnFail	Specifies whether GeocodeUSAddress normalizes an addresses that fail to match, and addresses that are unchanged. Normalization formats an address to the USPS guidelines without validating the address. One of the following:	
	Y	Perform standardization (default).
	N	Do not perform standardization.

optionName	Description / Valid Values
OutputPostalCodeSeparator	Specifies whether GeocodeUSAddress should include the dash in full postal code output. One of the following: **Y** Include the dash (default). **N** Do not include the dash.
OutputVerbose	Specifies whether GeocodeUSAddress provides an additional description field as output. These fields provide the text equivalent to a field represented by a code. For example, LocationCode returns a code that indicates the accuracy (quality) of the assigned geocode. LocationCode.Description provides the description for the code returned. One of the following: **Y** Include verbose fields. **N** Do not include verbose fields (default).

Output Data

The following table shows the GeocodeUSAddress options that control which data GeocodeUSAddress returns in the output.

Table 126: GeocodeUSAddress Output Data Options

optionName	Description / Valid Values
OutputRecordType	Specifies optional data to include in the output. Note that GeocodeUSAddress always returns the default data listed in **Default Output** on page 492. The data you select here is returned with the default output data. • **B**—Block Address • **C**—Census • **D**—DPV (The DPV field group is disabled unless you select **Perform Delivery Point Validation (DPV)** on the **Configuration Options** tab.) • **Z**—Geoconfidence • **L**—Latitude/Longitude • **E**—Parsed Elements • **P**—Postal Data • **Q**—Qualifiers • **R**—Range • **S**—Segment

optionName	Description / Valid Values
	For a description of the fields in each output group, see **Output** on page 483. If you do not want all of the fields in a particular category returned, do not select the check box, and list only those fields you want returned in OutputFields.
OutputFields	Specifies the individual output fields you want returned. List fields with a pipe (|) between each field. You can use this option instead of the OutputRecordType option to limit the output to those fields that are important to your data needs. By default, these are the address fields returned: AddressLine1|LastLine|Longitude|Latitude|MatchCode|LocationCode For a list of all the fields included in each data field, see **Output** on page 483.

Output

GeocodeUSAddress always returns a default set of output fields that contain the latitude/longitude, standardized address, and result indicators. For information on these fields, see **Default Output** on page 492. You can also choose to include the following optional categories of output data:

- **Auxiliary** on page 483
- **Block Address** on page 484
- **Census** on page 485
- **Centerline Projection** on page 489
- **Default Output** on page 492
- **DPV** on page 495
- **Geoconfidence** on page 497
- **Latitude/Longitude** on page 498
- **Parsed Elements** on page 498
- **Postal Data** on page 500
- **Qualifiers** on page 503
- **Range** on page 506
- **Segment** on page 507

Note: The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Auxiliary

Auxiliary data output fields contain information on the an auxiliary file match. For more information on using an auxiliary file, see **Auxiliary File Overview** on page 568. GeocodeUSAddress only returns values when matching against an auxiliary file. To include segment data fields in the output, set **OutputRecordType** = X.

Table 127: Auxiliary Data Output Fields

columnName	Description / Valid Values
AuxiliaryData	The user data field in an auxiliary file match. **Note:** GeocodeUSAddress does not process this information. It simply includes the user data contained in the auxiliary file.
MCDCode	The Minor Civil Division (MCD) code. A Minor Civil Division is a subdivision of a county, such as a township. There are Minor Civil Divisions in 28 states, the District of Columbia, Puerto Rico, and Island Areas. Minor Civil Divisions are defined by U.S. Census Bureau.
MCDName	The Minor Civil Division (MCD) name. A Minor Civil Division is a subdivision of a county, such as a township. There are Minor Civil Divisions in 28 states, the District of Columbia, Puerto Rico, and Island Areas. Minor Civil Divisions are defined by U.S. Census Bureau.

Block Address

Block data output fields contain extraneous information from the input address that GeocodeUSAddress could not process. To include block data in the output, set **OutputRecordType** = B.

If there are any empty lines in the input fields AddressLine1 through AddressLine6, GeocodeUSAddress moves the output lines to the first empty BlockLine output field, eliminating the blank lines. For example:

Input Field	Input Data	Output Field	Output Data
		AddressLine1	4750 Walnut St Ste 200
		LastLine	Boulder, CO 80301-2532
AddressLine1	Pitney Bowes	BlockLine1	Pitney Bowes
AddressLine2	4750 Walnut	BlockLine2	
AddressLine3		BlockLine3	
AddressLine4	Ste 200	BlockLine4	Dept ABC **Note:** Moved up one line from the input AddressLine5.

Input Field	Input Data	Output Field	Output Data
AddressLine5	Dept ABC	BlockLine5	
AddressLine6	80301	BlockLine6	

Table 128: Block Data Output Fields

columnName	Description
BlockLine1	Returns input address information GeocodeUSAddress could not process.
BlockLine2	Returns input address information GeocodeUSAddress could not process.
BlockLine3	Returns input address information GeocodeUSAddress could not process.
BlockLine4	Returns input address information GeocodeUSAddress could not process.
BlockLine5	Returns input address information GeocodeUSAddress could not process.
BlockLine6	Returns input address information GeocodeUSAddress could not process.

Census

Census output fields contain U.S. Census information about the address. To include census data in the output, set **OutputRecordType** = C.

Table 129: Census Data Output Fields

columnName	Description
APN	The assessor's parcel number of the property. The assessor's parcel number is an ID number assigned to a property by the local property tax authority.
BlockSuffix	The block suffix for the Census block in which the address is located.

columnName	Description
	A block suffix is a single character assigned to subsections of U.S. Census blocks that are split by a higher-level boundary, such as a municipal boundary. A block suffix is either "A" or "B". For information about U.S. Census block suffixes, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** Block suffixes are only available if you are using Centrus Enhanced data.
CBSACode	The code for the Core Based Statistical Area (CBSA) in which the address is located. A CBSA is a collective term that refers to both metropolitan and micropolitan areas. A metropolitan area has a population of more than 50,000, and a micropolitan area has a population between 10,000 and 49,999. For more information, see *Metropolitan and Micropolitan Statistical Areas* section of the U.S. Census Bureau website: **www.census.gov/population/www/metroareas/metroarea.html**
CBSADivisionCode	The code of the Core Based Statistical Area (CBSA) division in which the address is located. A CBSA division is a metropolitan statistical area with a population of at least 2.5 million that has been subdivided to form smaller groupings of counties referred to as "metropolitan divisions." For more information, see *Metropolitan and Micropolitan Statistical Areas* section of the U.S. Census Bureau website: **www.census.gov/population/www/metroareas/metroarea.html**
CBSADivisionName	The name of the Core Based Statistical Area (CBSA) division in which the address is located. A CBSA division is a metropolitan statistical area with a population of at least 2.5 million that has been subdivided to form smaller groupings of counties referred to as "metropolitan divisions." For more information, see *Metropolitan and Micropolitan Statistical Areas* section of the U.S. Census Bureau website: **www.census.gov/population/www/metroareas/metroarea.html** The CBSA division name is only returned if you set the option OutputVerbose=Y.

columnName	Description
CBSAMetro	Indicates whether the core based statistical area (CBSA) in which the address is located is a metropolitan area or a micropolitan area. One of the following: **Y** Yes, the address is located in a metropolitan statistical area. Metropolitan areas have a population greater than 50,000. **N** No, the address is not located in a metropolitan area. It is located in a micropolitan area. Micropolitan areas have a population between 10,000 and 49,999. **null** There is no data available to determine whether the address is in a metropolitan or micropolitan area. A CBSA is a collective term that refers to both metropolitan and micropolitan areas. A metropolitan area has a population of more than 50,000, and a micropolitan area has a population between 10,000 and 49,999. For more information, see *Metropolitan and Micropolitan Statistical Areas* section of the U.S. Census Bureau website: **www.census.gov/population/www/metroareas/metroarea.html**
CBSAName	The name of the core based statistical area (CBSA) in which the address is located. The CBSA division name is only returned if you set the option OutputVerbose=Y. A CBSA is a collective term that refers to both metropolitan and micropolitan areas. A metropolitan area has a population of more than 50,000, and a micropolitan area has a population between 10,000 and 49,999. For more information, see *Metropolitan and Micropolitan Statistical Areas* section of the U.S. Census Bureau website: **www.census.gov/population/www/metroareas/metroarea.html**
CensusBlockID	The 15-digit identification number of the census block in which the address is located. Census blocks are the smallest geographic area for which the Bureau of the Census collects and tabulates decennial census data. Census blocks are formed by streets, roads, railroads, streams and other bodies of water, other visible physical and cultural features, and the legal boundaries shown on Census Bureau maps. For more information about U.S. Census blocks, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** The Census block ID is in the format: `ssccctttttgbbb` Where:

columnName	Description
	ss The two-digit state FIPS code. **ccc** The three-digit county FIPS code. **tttttt** The six-digit Census tract FIPS code. **g** The single-digit block group FIPS code. **bbb** The block FIPS code. **Note:** GeocodeUSAddress does not return a period for the Centrus tract FIPS code. This may deviate from the industry standard.
CensusTract	The six-digit ID of the Census tract in which the address is located. Census tracts are small, relatively permanent geographic entities within counties (or the statistical equivalents of counties). Generally, census tracts have between 2,500 and 8,000 residents and boundaries that follow visible features. For more information about U.S. Census tracts, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html**
CSACode	Denotes the code for a geographic entity that consists of 2 or more adjacent CBSAs with employment interchange measures of at least 15.
CSAName	The name of the combined statistical area (CSA) in which the address is located. A CSA is a combination of two or more adjacent Core Based Statistical Areas (CBSAs) with a high employment interchange measure. The employment interchange measure is the sum of the percentage of employed residents of the smaller entity who work in the larger entity and the percentage of the employment in the smaller entity that is accounted for by workers who reside in the larger entity. Pairs of CBSAs with employment interchange measures of at least 25% combine automatically. Pairs of CBSAs with employment interchange measures of at least 15%, but less than 25%, may combine if local opinion in both areas favors combination. The CSA name is only returned if you set the option OutputVerbose=Y.
USCountyName	The name of the county or parish in which the address is located. The county/parish name is only returned if you set the option OutputVerbose=Y.

columnName	Description
USFIPSCountyNumber	The three-digit FIPS county code of the county in which the address is located.
USFIPSStateCode	The two-digit FIPS state code of the state in which the address is located.
USFIPSStateCountyCode	The five-digit FIPS code for the state and county in which the address is located.

Centerline Projection

Centerline projection output fields contain information specific to a centerline match. To include centerline projection fields in the output, set **OutputRecordType** = N.

Table 130: Centerline Projection Output Fields

columnName	Description
CenterlineBearing	The compass direction, in decimal degrees, from the point data match to the street centerline match. The compass direction is measured clockwise from 0 degrees north. For example, if the centerline match is directly north of the point match, the centerline bearing would be 0.
CenterlineBlockLeft	The Census FIPS Code that indicates the address is on the left side of the street.
CenterlineBlockRight	The Census FIPS Code that indicates the address is on the right side of the street.
CenterlineBlockSuffixLeft	The block suffix of the block on the left side of the street. A block suffix is a single character assigned to subsections of U.S. Census blocks that are split by a higher-level boundary, such as a municipal boundary. A block suffix is either "A" or "B". For information about U.S. Census block suffixes, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** Block suffixes are only available if you are using Centrus Enhanced data.
CenterlineBlockSuffixRight	The block suffix of the block on the right side of the street.

columnName	Description
	A block suffix is a single character assigned to subsections of U.S. Census blocks that are split by a higher-level boundary, such as a municipal boundary. A block suffix is either "A" or "B". For information about U.S. Census block suffixes, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** Block suffixes are only available if you are using Centrus Enhanced data.
CenterlineDataCode	Indicates the data used to obtain a centerline match for the address. One of the following: **0** USPS data in either the Centrus Enhanced, Centrus GDT, or Centrus NAVTEQ database. **1** TIGER data in the Centrus Enhanced database. **2** TeleAtlas data in the Centrus GDT database. **6** NAVTEQ data in the Centrus NAVTEQ database. **7** TeleAtlas point-level data in the Centrus TeleAtlas Points database. **8** Point-level data from the Centrus Points database. **9** Auxiliary file data. For more information about these databases, see **Enterprise Geocoding Databases** on page 294.
CenterlineDirection	Indicates the order of numbers on a segment for a centerline match. **F** Forward **R** Reversed **B** Both **U** Undetermined
CenterlineDistance	Distance, in feet, from the point-level match to the centerline match.
CenterlineHouseNumberHigh	The highest address number in the range of addresses on the street segment. For example, if the address range for the street segment is 1000 to 2000, the CenterlineHouseNumberHigh would be 2000.

columnName	Description
CenterlineHouseNumberLow	The lowest address number in the range of addresses on the street segment. For example, if the address range for the street segment is 1000 to 2000, the CenterlineHouseNumberLow would be 1000.
CenterlineIsAlias	Three characters indicating that GeocodeUSAddress located a centerline match by an index alias. The first is an `N` for normal street match or `A` for alias match (including buildings, aliases, firms, etc.). The next two characters are: **01** Basic index (normal address match) **02** USPS street name alias index **03** USPS building index **04** USPS firm name index **05** Statewide intersection alias match (when using the Usw.gsi or Use.gsi file) **06** Spatial data street name alias (when using, the Us_pw.gsi, Us_pe.gsi, Us_psw.gsi, or Us_pse.gsi file is required) **07** Alternate index (when using Zip9.gsu, Zip9e.gsu, and Zip9w.gsu) **08** LACS^{Link}
CenterlineLatitude	A seven-digit number in degrees and calculated to four decimal places for a centerline match. This field is only returned if AlwaysFindCandidates=Y
CenterlineLeadingDirectional	The street directional that precedes the street name for a centerline match. For example, the N in 138 N Main Street.
CenterlineLongitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified) for a centerline match. This field is only returned if AlwaysFindCandidates=Y
CenterlineParity	Indicates which side of the street has odd numbers for a centerline match. **L** The left side of the street has odd numbers. **R** The right side of the street has odd numbers. **B** Both sides of the street have odd numbers.

columnName	Description
	U Undetermined.
CenterlineRoadClass	The type of road for a centerline match: **1** Major **2** Minor
CenterlineSegmentCode	The unique 10-digit street segment ID assigned by the street network data provider.
CenterlineStreetName	The name of the street.
CenterlineStreetSuffix	The street type of the matched centerline location. For example, AVE in "Washington AVE".
CenterlineTrailingDirectional	The street directional that follows the street name. For example, the N in 456 Washington AVE N.

Default Output

GeocodeUSAddress always returns fields that contain the latitude/longitude, standardized address, and result indicators. Result indicators describe how well GeocodeUSAddress matched the input address to a known address and assigned a location. They also describes the overall status of a match attempt.

Table 131: Default Output Fields

columnName	Description
AddressLine1	The first line of the address. For example: **1 Global View** Troy, NY 12180-8371
AddressLine2	The second line of the address. For example: 4200 Parliament Pl **STE 600** Lanham, MD 20706-1882
City	The municipality name.

columnName	Description

Confidence	Indicates the confidence in the output provided, from 0 to 100. The higher the score, the higher the probability that the match is correct. If the match is exact, the confidence score is 100. For all other matches, GeocodeUSAddress calculates the confidence score by subtracting values from 100 as follows:
	• If GeocodeUSAddress changed the state to obtain a match:
	  • Added the state -3.75
	  • No state -7.5
	• If GeocodeUSAddress changed the city to obtain a match:
	  • Added city -2.5
	  • No city -5.0
	• If GeocodeUSAddress change the house number to obtain a match:
	  • Added house number -3.75
	  • No house number -7.5
	• If GeocodeUSAddress changed the street name to obtain a match:
	  • Added street name -3.75
	  • No street name -7.5
	• If GeocodeUSAddress changed the trailing directional to obtain a match:
	  • Added trailing directional -1.25
	  • No trailing directional -2.5
	• If GeocodeUSAddress changed the leading directional to obtain a match:
	  • Added leading directional -1.25
	  • No leading directional -2.5
	• If GeocodeUSAddress changed the street suffix to obtain a match:
	  • Added street suffix -1.25
	  • No street suffix -2.5
	• If GeocodeUSAddress changed the postal code to obtain a match: -11.25
	If you have enabled the option to return centroids, the confidence value indicates the type of centroid returned:
	• 60 for a street centroid

columnName	Description
	• 50 for a postal code centroid • 35 for a city centroid • 30 for a county centroid • 25 for a state centroid
Country	The name of the country. Since GeocodeUSAddress only works for U.S. locations, this field will always contain **United States of America**.
FirmName	The name of the business if the address is a business address.
LastLine	The complete last address line (city, state, and postal code).
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
LocationCode	A value indicating the accuracy (quality) of the assigned geocode. For more information, see **Address Location Codes** on page 575.
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
MatchCode	Indicates the portions of the address that matched to the directory file. For more information, see **Geocoding Match Codes** on page 592.
PostalCode	Nine-digit ZIP Code with or without a hyphen.
PostalCode.AddOn	Four-digit ZIP Code extension
PostalCode.Base	Five-digit ZIP Code.
ProcessedBy	The underlying software that processed the request. KGL for GeocodeUSAddress.
StateProvince	Two-character state abbreviation.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure

columnName	Description
Status.Code	If GeocodeUSAddress could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data
Status.Description	If GeocodeUSAddress could not process the address, this field will show a description of the failure. **Problem + explanation** Returned when Status.Code = Internal System Error. **Geocoding Failed** Returned when Status.code = No Geocode Found. **No location returned** Returned when Status.code = No Geocode Found.
StreetDataType	The data set GeocodeUSAddress attempted to match against.
StreetSide	Indicates the side of the street the range occupies. One of the following: **L** The range occupies the left side of the street. **R** The range occupies the right side of the street. **B** The range occupies both sides of the street. **U** Undetermined.
USUrbanName	Urbanization name. Used for addresses in Puerto Rico.

DPV

DPV data output fields contain information about a match made using DPV data. GeocodeUSAddress only returns values when matching against DPV data. To include DPV data in the output, set **OutputRecordType** = D.

Table 132: DPV Data Output Fields

columnName	Description
CMRA	Indicates whether the address is for a Commercial Mail Receiving Agent (CMRA). A CMRA is a private company that rents out mailboxes. A

columnName	Description
	customer of a commercial mail receiving agency can receive mail and other deliveries at the street address of the CMRA rather than the customer's own street address. Depending on the agreement between the customer and the CMRA, the CMRA can forward the mail to the customer or hold it for pickup. **Y** Yes, the address is a CMRA. **N** No, the address is not a CMRA. **null** DPV data is not available. DPV data is required to determine if an address is a CMRA.
DPV	Indicates whether the address is confirmed to be a deliverable address by USPS Delivery Point Validation (DPV). **N** Nothing confirmed **Y** Everything confirmed (ZIP+4, primary, and secondary) **S** ZIP+4 and primary (house number) confirmed **D** ZIP+4 and primary (house number) confirmed and a default match **U** Non-matched input address to USPS ZIP+4 data, or DPV data not loaded

DPVFootnote	Contains detailed information about the address. The DPV footnote codes are combined together consecutively.
	DPV footnotes include the following:
	• FOOTNOTE1 provides information about matched DPV records.
	  • **AA**—ZIP+4 matched record
	  • **A1**—Failure to match a ZIP+4 record
	  • **null**—Address not presented to hash table or DPV data not loaded
	• FOOTNOTE2 provides information about matched DPV records.
	  • **BB**—All DPV categories matched
	  • **CC**—Matched primary/house number, where the secondary/unit number did not match (present but invalid)
	  • **M1**—Missing primary/house number
	  • **M3**—Invalid primary/house number
	  • **N1**—Matched primary/house number, with a missing highrise secondary number
	  • **P1**—Missing PS, RR, or HC Box number

columnName	Description
	• **P3**—Invalid PS, RR or HC Box number • **F1**—All military addresses • **G1**—All general delivery addresses • **U1**—All unique ZIP Code addresses • **null**—Address not presented to hash table or DPV data not loaded • FOOTNOTE3 provides information about matched DPV records. • **R1**—Matched CMRA, without a present secondary/unit number • **RR**—Matched CMRA • **null**—Address not presented to hash table or DPV data not loaded **Note:** A unique ZIP Code is a ZIP Code assigned to a company, agency, or entity with sufficient mail volume to receive its own ZIP Code.

Geoconfidence

Geoconfidence data output fields contain information about the type of geoconfidence polygon returned. To include geoconfidence fields in the output, set OutputRecordType = Z.

columnName	Description
GeoConfidenceCode	The value returned in this field indicates which geoconfidence surface type has been returned. Possible values are: **INTERSECTION** A geocode point for the intersection of two streets. **ADDRESS** An array of street segment points representing the street segment where the address is located. **POINT** If the geocoder was able to match the address using point data, the point geometry where the address is located. **POSTAL1** A geocode point for the ZIP centroid. **POSTAL2** An array of points for all street segments in the ZIP + 2 in which the address is located.

columnName	Description
	POSTAL3 An array of points for street segments in the ZIP + 4 in which the address is located. **ERROR** An error has occurred.
StreetSegmentPoints	An array of latitude/longitude values that represent the street segment points. **Note:** This field contains values only if the `GeoConfidenceCode` field returns a value of `ADDRESS`, `POSTAL2`, or `POSTAL3`.
GeoConfidenceCentroidLatitude	The latitude of the centroid of the geoconfidence polygon.
GeoConfidenceCentroidLongitude	The longitude of the centroid of the geoconfidence polygon.

Latitude/Longitude

The latitude/longitude output fields contain the geographic coordinates of the address. To include latitude/longitude output fields in the output, set **OutputRecordType** = L.

Table 133: Latitude/Longitude Output Fields

columnName	Description
Elevation	The location's elevation in feet above or below sea level.
Latitude	The latitude of the address. The latitude is a seven-digit number in degrees, calculated to six decimal places.
Longitude	The longitude of the address. The longitude is a seven-digit number in degrees, calculated to six decimal places.

Parsed Elements

The parsed elements output fields contain standard address information as individual units, such as street suffixes (for example AVE, ST, or RD) and leading directionals (for example N and SE). To include parsed elements in the output, set **OutputRecordType** = E.

Table 134: Parsed Elements Output Fields

columnName	Description
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentLabel2	The type of unit, such as apartment, suite, or lot, for addresses that contain two units, such as: 123 E Main St APT 3, 4th **Floor** .
ApartmentNumber	Apartment number. For example: 123 E Main St APT **3**
ApartmentNumber2	Secondary apartment number. For example: 123 E Main St APT 3, **4th** Floor
CrossStreetLeadingDirectional[1]	Leading directional, for example: 123 **E** Main St Apt 3
CrossStreetName	Name of cross street.
CrossStreetSuffix[1]	Street suffix, for example: 123 E Main **St** Apt 3
CrossStreetTrailingDirectional[1]	Trailing directional, for example: 123 Pennsylvania Ave **NW**
HouseNumber	Building number for the address.
HouseNumber2	If an address consists of a range of house numbers, this field contains the second house number. The HouseNumber field contains the first number. For example, given this address: 5-7 Maple Ave. The HouseNumber field would contain "5" and the HouseNumber2 field would contain "7".
LeadingDirectional	Leading directional, for example: 123 **E** Main St Apt 3
PrivateMailbox	Private mailbox. Not returned for multiline input.
PrivateMailbox.Designator	Private mailbox description. Not returned for multiline input.
RRHC	Rural Route/Highway Contract portion of the address.
StreetName	Street name.

[1] GeocodeUSAddress only returns Cross street outputs if you entered an intersection as an address. For example, entering Pearl and 28th, Boulder, CO returns cross street information. Entering 2800 Pearl, Boulder, CO does NOT return cross street information.

columnName	Description
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 Washington N.

Postal Data

Postal data output fields contain detailed postal information for the address, such as the preferred city name and the US carrier route. To include postal data fields in the output, set **OutputRecordType** = P.

Table 135: Postal Data Output Fields

columnName	Description
CityPreferredName	The USPS® preferred city name for the ZIP Code of the address.
CityShortName	The USPS®-approved abbreviation for the city, if there is one. The USPS® provides abbreviations for city names that are 14 characters long or longer. City abbreviations are 13 characters or less and can be used when there is limited space on the mailing label. If there is no short city name for the city, then the full city name is returned.
CityStateRecordName	USPS® city state city name.
DeliveryPointCode	Two-digit delivery point bar code.
GovernmentBuilding	Indicates if a building is used by the city, state, or federal government. **A** City government building **B** Federal government building **C** State government building **D** Firm only **E** City government building and firm only **F** Federal government building and firm only **G** State government building and firm only The values A, B, C, E, F, and G are valid for Alternate records only. The value D is valid for both base and alternate records.

columnName	Description
PostalBarCode	Six-digit combination of ZIP+4 Code and the delivery point bar code.
PostalCodeClass	ZIP Classification code. **null** Standard ZIP Code **M** Military ZIP Code **P** ZIP Code has P.O. boxes only **U** Unique ZIP Code (ZIP Code assigned to a single organization)
PostalCodeUnique	Indicates if the ZIP Code is a unique ZIP Code assigned to an individual company or agency. **Y** Unique ZIP name **null** No unique ZIP name
PostalFacility	USPS City State Name Facility code. **A** Airport Mail Facility (AMF) **B** Branch **C** Community Post Office (CPO) **D** Area Distribution Center (ADC) **E** Sectional Center Facility (SCF) **F** Delivery Distribution Center (DDC) **G** General Mail Facility (GMF) **K** Bulk Mail Center (BMC) **M** Money Order Unit **N** Non-Postal Community Name, Former Postal Facility, or Place Name **P** Post Office **S** Station **U** Urbanization
USBCCheckDigit	Check-digit for delivery point bar code.
USCarrierRouteCode	Carrier Route code.

columnName	Description
USCarrierRouteSort	Indicates if the USPS uses a carrier route sort, and what type of sort the USPS allows. **A** Automation cart allowed, optional cart merging allowed **B** Automation cart allowed, no optional cart merging allowed **C** No automation cart allowed, optional cart merging allowed **D** No automation cart allowed, no optional cart merging allowed
USCityDelivery	Indicates if has city-delivery carrier routes. **Y** Has city-delivery carrier routes **N** Does not have city-delivery carrier routes.
USLACS	Indicates if LACS^{Link} match occurred. **Y** Matched LACS^{Link} record **N** LACS^{Link} match not found **F** False-positive LACS^{Link} record **S** Secondary information (unit number) removed to make a LACS^{Link} match **null** Records not processed through LACS^{Link} For more information, see **Locatable Address Conversion System (LACS)** on page 303.
USLACS.ReturnCode	Indicates LACS^{Link} results. **A** Matched LACS^{Link} record **00** LACS^{Link} match was not found **09** Matched to highrise default, but noLACS^{Link} conversion **14** Found LACS^{Link} match, but no LACS^{Link} conversion **92** Secondary information (unit number) was removed to make a LACS^{Link} match **null** Records not processed through LACS^{Link} For more information, see **Locatable Address Conversion System (LACS)** on page 303.

columnName	Description
USLOTCode	A combination of the 4-digit Line of Travel (LOT) Code and the ascending (A) or descending (D) indicator.

Qualifiers

Qualifier output fields contain qualification information on the match, such as the location code and the match code. To include postal data fields in the output, set OutputRecordType = Q.

Table 136: Qualifier Output Fields

columnName	Description
AddressLineResolved	For two-line addresses, indicates which address line was used to obtain the match. One of the following: **0** The address could not be matched, or the address matched to multiple addresses. **1** AddressLine1 was used to obtain the match. **2** AddressLine2 was used to obtain the match. **3** Both address lines were used in their original order. **4** Both address lines were used but the order of the lines was switched to obtain the match. **5** The input address was a one line address.
CountryLevel	The category of postal data available. Always returns A in GeocodeUSAddress—Validates, corrects, and provides missing postal code, city name, state/county name, street address elements, and country name.
DatabaseVersion	USPS publish date, in the format `Month Year`.
EWSMatch	Indicates if GeocodeUSAddress denied a match because of the input address matched an address in the Early Warning System (EWS) data. **Y** The address matched to an address in the EWS data so the match was denied. **null** The address did not match to an address in the EWS data.
ExpirationDate	Date the database expires, in the format `MM/DD/YY`.

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. **Note:** The match codes returned in this field are different from the match codes described in **Geocoding Match Codes** on page 592. Instead, the match codes returned in this field are taken from a set of match codes that are compatible with all other country geocoders. For more information, see **Introduction** on page 597.
GeoStanMatchScore	Record matching score (for multimatches only).
Intersection	Indicates if GeocodeUSAddress found a cross-street match. **T** True **F** False
IsAlias	GeocodeUSAddress located a match by an index alias. Returns 3 characters. The first is an `N` for normal street match or `A` for alias match (including buildings, aliases, firms, etc.). The next 2 characters are: **01** Basic index (normal address match) **02** USPS street name alias index **03** USPS building index **04** USPS firm name index **05** Statewide intersection alias match **06** Spatial data street name alias **07** Alternate index **08** LACS^{Link}
IsCloseMatch	Indicates whether or not the address is considered a close match. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
LACSAddress	Indicates if GeocodeUSAddress converted an address due to the Locatable Address Conversion System (LACS) **L** Converted

columnName	Description
	null Not converted
LocationCode.Description	LocationCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).
MatchCode.Description	MatchCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).
RecordType	Indicates the record type: • GeneralDelivery • HighRise • FirmRecord • Normal • PostOfficeBox • RRHighwayContract • Geographic (non USPS TIGER match) • Auxiliary (match to an auxiliary file)
RecordType.Default	Indicates type of match that occurred for the record type HighRise or RRHighwayContract: **Y** Default match **N** Exact match **U** Not matched
StreetDataCode	Indicates the data used to geocode the address. One of the following: **0** USPS data in either the Centrus Enhanced, Centrus GDT, or Centrus NAVTEQ database. **1** TIGER data in the Centrus Enhanced database. **2** TeleAtlas data in the Centrus GDT database. **6** NAVTEQ data in the Centrus NAVTEQ database. **7** TeleAtlas point-level data in the Centrus TeleAtlas Points database. **8** Point-level data from the Centrus Points database. **9** Auxiliary file data.

columnName	Description
	For more information about these databases, see **Enterprise Geocoding Databases** on page 294.
StreetDataType	Indicates the data initially used for the match attempt. Note that the output field StreetDataCode shows which data was actually used to obtain the match. The data indicated in StreetDataType may be different from that in StreetDataCode if a match cannot be made in the initial match attempt. For example, if a points database is loaded, GeocodeUSAddress will first attempt a match to the point data because this is the most accurate type of match. If a point-level match cannot be made, GeocodeUSAddress will attempt to match to street data. If the match is made using street data, then the SreetDataType would indicate the point-level data and the StreetDataCode would indicate the street data. For more information, see **How GeocodeUSAddress Processes Addresses** on page 466.

Range

Range output fields contain information on the street range, such as the high and low unit numbers. To include range data fields in the output, set **OutputRecordType** = R.

Table 137: Range Data Output Fields

columnName	Description
Alternate	USPS code that specifies whether a record is a base or alternate record. **B** Base record. Base records can represent a range of addresses or an individual address, such as a firm record. **A** Alternate record. Alternate records are individual delivery points.
HouseNumberHigh	The highest house number in the range.
HouseNumberLow	The lowest house number in the range.
HouseNumberParity	Indicates if the house number range contains even or odd numbers. **E** Even **O** Odd

columnName	Description
	B Both
PostalCodeExtensionHigh	The highest four-digit ZIP Code extension in the range. The ZIP Code extension is the four digits at the end of the ZIP Code. For example: 60510-**1134**.
PostalCodeExtensionLow	The lowest four-digit ZIP Code extension in the range. The ZIP Code extension is the four digits at the end of the ZIP Code. For example: 60510-**1134**.
UnitNumberHigh	The highest unit number in the range.
UnitNumberLow	The lowest unit number in the range.
UnitNumberParity	Indicates if the unit number range contains even or odd numbers. **E** Even **O** Odd **B** Both

Segment

Segment output fields contain information on the street segment identified by the data provider. To include segment data fields in the output, set **OutputRecordType** = S.

Table 138: Segment Data Output Fields

columnName	Description
BlockLeft	The Census FIPS Code that indicates the address is on the left side of the street.
BlockRight	The Census FIPS Code that indicates the address is on the right side of the street.
BlockSuffixLeft	The block suffix of the block on the left side of the street. A block suffix is a single character assigned to subsections of U.S. Census blocks that are split by a higher-level boundary, such as a municipal boundary. A block suffix is either "A" or "B". For information

columnName	Description
	about U.S. Census block suffixes, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** Block suffixes are only available if you are using Centrus Enhanced data.
BlockSuffixRight	The block suffix of the block on the right side of the street. A block suffix is a single character assigned to subsections of U.S. Census blocks that are split by a higher-level boundary, such as a municipal boundary. A block suffix is either "A" or "B". For information about U.S. Census block suffixes, see the *Geographic Areas Reference Manual*, available at the U.S. Census Bureau website: **www.census.gov/geo/www/garm.html** Block suffixes are only available if you are using Centrus Enhanced data.
RoadClass	The type of road: **1** Major **2** Minor
SegmentCode	The unique 10-digit street segment ID assigned by the street network data provider.
SegmentDirection	Indicates the order of numbers on a segment. **F** Forward **R** Reversed **B** Both **U** Undetermined
SegmentHouseNumberHigh	The highest house number in the segment.
SegmentHouseNumberLow	The lowest house number in the segment.
SegmentLength	The length, in feet, of a block segment.
SegmentParity	Indicates which side of the street has odd numbers.

columnName	Description	
	L	Left side of the street
	R	Right side of the street
	B	Both sides of the street
	U	Undetermined
StreetSide.NAVTEQ	Indicates which side of the street the address is located on. The value in this field is determined by using the NAVTEQ reference nodes for the street segment. A street segment represents part of a street. Each segment has a node at each end: the reference node and the non-reference node. The reference node is the node with the lower latitude (southernmost). If the latitudes of both nodes are identical, the reference node is the end node with the lower longitude (westernmost). The street side corresponds to the street sides you would see if you were standing at the reference node and looking at the non-reference node. One of the follwoing:	
	L	The address is on the left side of the street.
	R	The address is on the right side of the street.
	B	The address occupies both sides of the street.
	U	The street side is unknown.
	null	NAVTEQ data was not used, or segment output data was not selected, or the address did not match a street segment (for example, the address was geocoded to a centroid).

Reports

There is one report available with GeocodeUSAddress: the GeocodeUSAddress Summary Report. To create the report, in Enterprise Designer, drag the **GeocodeUSAddress Summary Report** icon to the canvas. You do not need to draw a connector to the report.

GeocodeUSAddress Summary Report

The GeocodeUSAddress Summary Report contains information about the job, such as the settings, number of records processed, performance statistics, and the database used, as well as detailed information about the geocoding and address matching results. This report contains the following sections.

Address Matching Summary

This section shows the number of records processed and the number of matches obtained.

- **Total Records in File**—The total number of records in the input file used by this job.
- **Records Processed**—The number of input records minus those records that were skipped.
- **Addresses Matched**—The number of addresses that were successfully standardized and validated. This count includes all the Standardized addresses listed under the Matched Address Records section plus those with match codes beginning with G (auxiliary file), T (geographic data only), and X (intersections).
- **Unmatched**—The number of records that could not be validated.

Matched Address Records

This section contains information about the addresses that were successfully matched.

- **Standardized**—The number of addresses that match to USPS-relevant records. These addresses have match codes beginning with A (alias), D (small town), Q (unique ZIP), S (street), and U (rare). Only these types of matches are counted as Standardized.
- **Auxiliary File**—The number of records that were matched to a user-defined file. These records have a MatchCode beginning with G.
- **Intersections**—The number of records that were matched to an intersection. These records have a MatchCode beginning with X.
- **Non-USPS**—The number of records that were matched to non-USPS data.

Unmatched Address Records

This section lists the number of unmatched addressees and the reasons why the addresses were not matched. For information on these codes, see **Location Codes for U.S. Geocoding** on page 574.

Standardized Address Quality

This section describes the changes that GeocodeUSAddress made to addresses in order to validate them.

- **Original address unchanged**—None of the address elements were changed to obtain a match.
- **Original last line unchanged**—The last line (city, state, ZIP Code) was unchanged but other elements were changed to obtain a match.
- **Corrected predirectional**—The predirectional of a street name was changed to obtain a match. For example, E MAIN ST changed to W MAIN ST.
- **Corrected street name**—The name of the street was changed to obtain a match. For example, MAIN ST changed to MAINE ST.
- **Corrected street suffix**—The street suffix was changed to obtain a match. For example, MAIN ST changed to MAIN AVE.
- **Corrected postdirectional**—The postdirectional of a street was changed to obtain a match. For example, MAIN ST NW changed to MAIN ST SW.
- **Corrected city name**—The name of the city was corrected to obtain a match. For example, LOS ANGLES changed to LOS ANGELES.
- **Corrected state abbreviation**—The state abbreviation was corrected to obtain a match. For example, ROCHESTER NY changed to ROCHESTER MN.
- **Corrected ZIP Code**—The ZIP Code as corrected to obtain a match. For example: 1071 MAPLE LN BATAVIA IL 49423 Changed to: 1071 MAPLE LN BATAVIA IL 60510

- **Corrected ZIP + 4 add on**—The four digits that appear after the "-" in a ZIP + 4 were corrected to obtain a match. For example, 60510 changed to 60510-1135.

Geocode Matching Levels

This section describes the level of accuracy for the geocodes returned by GeocodeUSAddress.

- **Geocodes Assigned**—The number of records to which GeocodeUSAddress assigned a geocode. This is the cumulative count of the following fields:
- **Address Match**—Address geocodes indicate a geocode made directly to a street network segment (or two segments, in the case of an intersection). Addresses included in this count have a value that begins with A in the **LocationCode** output field.
- **Auxiliary Match**—The geocode was determined using the Auxiliary File. Addresses included in this count have a value that begins with AG in the **LocationCode** output field. For more information, see **Auxiliary Match Details** on page 511.
- **Point Match**—The geocode was determined using a points database, which means the geocode represents the center of a building or parcel. Addresses included in this count have a value that begins with AP in the **LocationCode** output field. For more information, see **Point Matching Details** on page 511.
- **ZIP Centroids Match**—The address could not be matched, so the geocode is the center of the address's ZIP Code. This is the least accurate geocode for a given address. Addresses included in this count have a value that begins with Z in the **LocationCode** output field. For more information, see **ZIP Centroid Matching Details** on page 512.

Auxiliary Match Details

This section describes the level of accuracy for the geocodes returned by GeocodeUSAddress that were obtained from the Auxiliary File. For more information, see **Auxiliary File Overview** on page 568.

These fields are ordered from the most accurate type of geocode to the least.

- **Point**—The geocode represents the center of a building or parcel. Addresses included in this count have a value of AG0 in the **LocationCode** output field.
- **Centerline**—The geocode represents the location of an address along a street segment. Addresses included in this count have a value of AG1 in the **LocationCode** output field.
- **Centerline with unknown street side**—The geocode represents the location of an address along a street segment but the side of the street where the address resides could not be determined. Addresses included in this count have a value of AG2 in the **LocationCode** output field.
- **Midpoint**—The geocode represents the midpoint of the street segment where the address resides. GeocodeUSAddress could not determine where on the street segment the address is located. Addresses included in this count have a value of AG3 in the **LocationCode** output field.

Point Matching Details

This section describes the types of point-level geocodes returned by GeocodeUSAddress. Point-level geocodes represent the center of a parcel or building.

- **Parcel Centroid**—The geocode represents the center of a parcel. Addresses included in this count have a value of AP02 in the **LocationCode** output field.

- **Field-collected GPS**—The geocode is determined using data collected by teams of field verification specialists who drive the roads of selected areas to verify and update the data. Addresses included in this count have a value of AP04 in the **LocationCode** output field.
- **Structure Centroid**—The geocode represents the center of an addressable building footprint. An addressable structure is typically a structure that receives mail or has telephone service. Addresses included in this count have a value of AP05 in the **LocationCode** output field.
- **Manual Frontage Midpoint**—The geocode represents the center of a the parcel's boundary with the street. These points are offset at a specific distance from the street centerline near the center of the side of the parcel that fronts the street segment. Street frontage points estimate address locations more accurately than do interpolated ranges. Addresses included in this count have a value of AP07 in the **LocationCode** output field.
- **Unknown Point-Level Geocode**—The type of geocode is not known.

ZIP Centroid Matching Details

This section describes the types of ZIP Code centroids and census centroids returned by GeocodeUSAddress.

- **Location Accuracy**—These fields describe the accuracy of the ZIP Code centroids.
- **ZIP + 4**—The centroid indicates the center of a ZIP + 4 code. This is the most accurate type of ZIP centroid. Addresses included in this count have a value of 9 in the third character of the value in the **LocationCode** output field.
- **ZIP + 2**—The centroid represents the center of a ZIP + 2 code. Addresses included in this count have a value of 7 in the third character of the value in the **LocationCode** output field.
- **ZIP Code**—The centroid represents the center of a five-digit ZIP Code. This is the least accurate type of ZIP centroid. Addresses included in this count have a value of 5 in the third character of the value in the **LocationCode** output field.
- **Census Accuracy**—These fields describe the accuracy of the census centroids.
- **Block Group**—The centroid represents the center of a block group. This is the most accurate type of census centroid. Addresses included in this count have a value of that begins with ZB in the **LocationCode** output field.
- **Census Tract**—The centroid represents the center of a census tract. Addresses included in this count have a value of that begins with ZT in the **LocationCode** output field.
- **County**—The centroid represents the center of a county. This is the least accurate type of census centroid. Addresses included in this count have a value of that begins with ZC in the **LocationCode** output field.

LACS/Link Statistics

This section describes the results of LACS/Link address processing. For information on LACS/Link, see **Locatable Address Conversion System (LACS)** on page 303.

- **Records processed by LACS/Link**—Addresses that were processed using LACS/Link.
- **LACS/Link Matched**—Addresses that were matched to addresses in the LACS/Link database.
- **LACS/Link Matched w/ dropped unit info**—Addresses whose secondary address information was removed in order to obtain a LACS^{Link} match.
- **Not LACS/Link Matched**—Addresses that GeocodeUSAddress attempted to match to LACS^{Link} but were not found in the LACS^{Link} database.

- **Not LACS/Link Converted**—The address matched a LACS^{Link} record but was not converted.
- **Not LACS/Link Converted - highrise default**—The address matched a highrise default record but was not converted.
- **Last LACS/Link false positive record**—This is the record number within the input file of the last address to result in a false positive. For example, if the 5th record in the file was a LACS^{Link} false positive, this field would contain "5". For more information on false positives, see **Encountering False Positives** on page 609.

Delivery Point Validation

This section describes the results of DPV address processing. For information on DPV, see **Delivery Point Validation (DPV)** on page 303.

- **Records processed by DPV**—The number of addresses that were processed using DPV.
- **DPV Records with ZIP + 4**—Addresses that contained a ZIP + 4 code and were processed by DPV.
- **DPV Confirmed**—The number of addresses that were verified as deliverable addresses.
- **Primary Confirmed, Secondary Missing**—The primary portion of the address (the house number and street) was verified. The address requires a secondary element (for example, a suite or apartment number) to be a deliverable address, and the secondary information was missing from the input address.
- **Primary Confirmed, Secondary Incorrect**—The primary portion of the address (the house number and street) were verified. The address requires a secondary element (for example, a suite or apartment number) to be a deliverable address, and the secondary information in the input address was incorrect.
- **DPV CMRA Confirmed**—Commercial Mail Receiving Agency (CMRA) addresses confirmed by DPV.
- **DPV Not Confirmed**—Addresses that could not be verified as deliverable.
- **USPS Street Records Confirmed**—Street addresses that were confirmed by DPV.
- **USPS General Delivery Records Confirmed**—DPV processing confirmed that the address accepts general delivery mail.
- **Records with confirmed CMRA**—Commercial Mail Receiving Agency (CMRA) addresses that were confirmed with DPV.
- **Records not confirmed CMRA**—Commercial Mail Receiving Agency (CMRA) addresses that could not be confirmed with DPV.
- **DPV False Positive Seed table hits**—Addresses that matched to DPV false positive records. For more information, see **Encountering False Positives** on page 609.

Records with DPV Footnote

This section lists the DPV footnote codes that were returned for the job. For an explanation of the DPV footnote codes, see **DPV** on page 495.

USPS Firm Records

This section describes the results of address validation for firm (business) addresses.

- **Confirmed**—GeocodeUSAddress confirmed that the address is a business address.
- **Confirmed with PMB presented**—GeocodeUSAddress confirmed that the address is a business address, and the business address contains a private mailbox (PMB).

- **Failed primary house number**—Business addresses that contained a primary house number which could not be confirmed.
- **Failed secondary unit number**—Business addresses that contained a secondary unit number which could not be confirmed.

USPS Highrise Records

This section describes the results of DPV processing for highrise addresses.

- **Confirmed**—Highrise addresses that were confirmed by DPV.
- **Confirmed with PMB presented**—Highrise addresses that contain a Private Mailbox (PMB) and were confirmed by DPV.
- **Conf. CMRA with/without PMB**—Highrise addresses that are also CMRA addresses, and that did or did not contain a Private Mailbox (PMB) address element.
- **Failed primary house number**—Highrise addresses that contained a primary house number which could not be confirmed.
- **Failed secondary unit number**—Highrise addresses that contained a secondary unit number which could not be confirmed.

USPS PO Box Records

This section describes the results of DPV processing for PO box addresses.

- **Confirmed**—PO Box addresses that were confirmed by DPV.
- **Failed primary box number**—PO Box addresses that contained a primary box number which could not be confirmed.

USPS Rural Route Records

This section describes the results of DPV processing for rural route addresses.

- **Confirmed**—Rural Route addresses that were confirmed by DPV.
- **Conf. CMRA with/without PMB**—Rural Route addresses that were also CMRA addresses, and that did or did not contain a Private Mailbox (PMB) address element.
- **PMB Presented**—Rural Route addresses that contained a Private Mailbox (PMB) address element.
- **Failed primary house number**—Rural Route addresses that contained a primary house number which could not be validated.

USPS Street Records

This section describes the results of DPV processing for street addresses.

- **Confirmed**—Street addresses that were confirmed by DPV.
- **Confirmed with PMB presented**—Street addresses that contained a Private Mailbox (PMB) and were confirmed by DPV.
- **Conf. CMRA with/without PMB**—Street addresses that were also CMRA addresses, and that did or did not contain a Private Mailbox (PMB) address element.
- **Failed primary house number**—Street addresses that contained a primary house number which could not be confirmed.

- **Failed secondary unit number**—Street addresses that contained a secondary unit number which could not be confirmed.

GNAFPIDLocationSearch

GNAFPIDLocationSearch identifies the address and latitude/longitude coordinates for a Geocoded National Address File Persistent Identifier (G-NAF PID). The G-NAF PID is a 14-character alphanumeric string that uniquely identifies each G-NAF address in the G-NAF database (a database of Australian locations). The PID is constructed from a combination of the major address fields of the G-NAF database. An example of a G-NAF PID is:

GAVIC411711441

Note: You must have the G-NAF database installed to use GNAFPIDLocationSearch.

GNAFPIDLocationSearch is part of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Input

GNAFPIDLocationSearch takes a G-NAF PID as input. The following table provides information on the format and layout of the input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 139: GNAFPIDLocationSearch Input

columnName	Format	Description/Valid Values
GNAFPID	String	The 14-character G-NAF persistent identifier you want to look up. For example: GAVIC411711441

Options

GNAFPIDLocationSearch allows you to set default processing options through the Management Console. You can override certain settings for individual calls to GNAFPIDLocationSearch using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

The following sections describe the options that control GNAFPIDLocationSearch processing.

Geocoding Options

Table 140: GNAFPIDLocationSearch Geocoding Options

optionName	Description/Valid Values
GNAFPointType	Specifies whether to return the parcel latitude/longitude or the street frontage latitude/longitude. This option is only available if you have the G-NAF database installed. This option only affects addresses matched to the G-NAF database. One of the following: **P** In a street address match, return the exact location of the parcel. This is the standard G-NAF point which is the exact authoritative point returned by the G-NAF database. Default. **S** In a street address match, return the street frontage point for the parcel. The street frontage point is 12.5 metres from the front boundary of the parcel. Street frontage points are more suitable for routing applications.
Return8DecimalPlaceLatLong	Specifies whether to return the original latitude and longitude, precise up to eight digits after the decimal. This is the latitude/longitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This option is only available if you have the G-NAF database installed. This option only affects addresses matched to the G-NAF database. One of the following: **Y** Yes, return the original latitude/longitude, up to eight digits after the decimal. **N** No, do not return the original latitude/longitude.

Data Options

Table 141: GNAFPIDLocationSearch Data Options

optionName	Description/Valid Values
Database	Specifies the database to use to look up the parcel. Use the database name specified in the Management Console's Database Resources

optionName	Description/Valid Values
	tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. **Note:** Only database resources that contain G-NAF databases are available in this list.

Output

GNAFPIDLocationSearch returns the following categories of data:

The following tables list the output fields included in each category.

Note: The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Address Output

Table 142: Address Output

columnName	Description
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
City	Municipality name.
Country	The three-letter ISO 3166-1 Alpha 3 country code.
County	The Local Government Authority (LGA).
FirmName	Name of the company or a place name.
HouseNumber	Building number for the matched location.
HouseNumberHigh	The highest house number of the range in which the address resides.
HouseNumberLow	The lowest house number of the range in which the address resides.

columnName	Description
HouseNumberParity	Indicates if the house number range contains even or odd numbers or both. **E** Even **O** Odd **B** Both
LastLine	Complete last address line (city, state/province, and postal code).
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
Locality	This field is not used for this country.
NumberOfCandidateRanges	Indicates whether or not the address has a house number. One of the following: **0** The address has no house number. Examples of addresses that have no house number are P.O. box addresses and general delivery addresses. **1** The address has a house number. For information about the range in which the house number falls, see the HouseNumberHigh, HouseNumberLow, and HouseNumberParity fields.
NumberOfRangeUnits	Indicates whether or not the address has a unit number, such as a suite number or apartment number. One of the following: **0** The address has no unit number. **1** The address has a unit number. For information about the range in which the unit number falls, see the UnitNumberHigh and UnitNumberLow.
PostalCode	The postcode for the address. The format of the postcode varies by country.
PostalCode.Addon	The second part of a postcode. For example, for Canadian addresses this will be the LDU. This field is not used by most countries.
PreAddress	Miscellaneous information that appears before the street name.
PrivateMailbox	This field is not currently used.
SegmentParity	Indicates which side of the street has odd numbers.

columnName	Description
	L Left side of the street **R** Right side of the street **B** Both sides of the street **U** Undetermined
StateProvince	The state name.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.
StreetName	The street name.
StreetPrefix	The type of street when the street type appears before the base street name. For example, AVENUE: 12 **AVENUE** B KALGOORLIE WA 6430
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 Washington N.
UnitNumberHigh	The highest unit number of the range in which the unit resides.
UnitNumberLow	The lowest unit number of the range in which the unit resides.

Geocode Output

Table 143: Geocode Output

columnName	Description
CoordinateSystem	The coordinate system used to determine the latitude and longitude coordinates. A coordinate system specifies a map projection, coordinate units, etc. An example is EPSG:4326. EPSG stands for European Petroleum Survey Group.

columnName	Description
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).

Result Codes

Result codes contain information about the success or failure of the geocoding attempt, as well as information about the accuracy of the geocode.

Table 144: Result Code Output

columnName	Description
Geocoder.MatchCode	Indicates how closely the input address matches the candidate address. For more information, see **Introduction** on page 597.
IsCloseMatch	Indicates whether or not the address is considered a close match. An address is considered close based on the "Close match criteria" options on the Matching tab. **Y** Yes, the address is a close match. **N** No, the address is not a close match.
MultiMatchCount	For street address geocoding, the number of matching address positions found for the specified address. For intersection geocoding, the number of matching street intersection positions found for the specified addresses.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	If the geocoder could not process the address, this field will show the reason. • Internal System Error • No Geocode Found • Insufficient Input Data

columnName	Description
	• Multiple Matches Found • Exception occurred • Unable to initialize Geocoder • No Match Found
Status.Description	If the geocoder could not process the address, this field will show a description of the failure. **Problem + explanation** — Returned when Status.Code = Internal System Error. **Geocoding Failed** — Returned when Status.code = No Geocode Found. **No location returned** — Returned when Status.code = No Geocode Found. **No Candidates Returned** — The geocoder could not identify any candidate matches for the address. **Multiple Candidates Returned and Keep Multiple Matches not selected** — The address resulted in multiple candidates. In order for the candidate address to be returned, you must specify `KeepMultimatch=Y`.
LocationPrecision	A code describing the precision of the geocode. One of the following: **0** — No coordinate information is available for this candidate address. **1** — Interpolated street address. **2** — Street segment midpoint. **3** — Postal code 1 centroid. **4** — Partial postal code 2 centroid. **5** — Postal code 2 centroid. **6** — Intersection. **7** — Point of interest. **8** — State/province centroid. **9** — County centroid. **10** — City centroid. **11** — Locality centroid.

columnName	Description
	12 Additional point precision for unspecified custom item. **13** Additional point precision for unspecified custom item. **14** Additional point precision for unspecified custom item. **15** Additional point precision for unspecified custom item. **16** The result is an Address Point. **17** The result was generated by using address point data to modify the candidates segment data.
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.

G-NAF Output

The following table lists output fields that are unique to the Australian Geocoded National Address File (G-NAF®) database. G-NAF is an optional database that is available for all six states and two territories. G-NAF is the only authoritative Australian national index of locality, street and number, validated with geographic coordinates.

Table 145: Austraila G-NAF Output

columnName	Description
AUS.GNAF_CONFIDENCE	A number indicating how many G-NAF datasets the address is found in. A higher confidence level means that the same address was found in more data contributor sources. One of the following: **<number>** The number of datasets the address was found in, minus 1. For example, a value of 0 indicates that the address was found in one contributor's dataset, a value of 1 indicates that the address was found in two contributors' datasets, a value of 2 indicates that the address was found in three contributors' datasets, and so forth.

columnName	Description
	-1 The address could not be found in any G-NAF dataset.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LATITUDE	The parcel latitude, precise to eight digits after the decimal. This is the latitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_EIGHT_DECIMAL_PLACE_LONGITUDE	The parcel longitude, precise to eight digits after the decimal. This is the longitude that the candidate matched to in the G-NAF database. These are the original coordinates directly from the G-NAF data prior to truncation or rounding. This field is only returned if you specify **Return8DecimalPlaceLatLong=Y**.
AUS.GNAF_GEOCODE_LEVEL	A number indicating the level o f geocode for the address. Every principal address within the G-NAF database has at least a locality level geocode. They may also have a street level geocode and a point level geocode. One of the following: **0** No geocode. **1** Parcel level geocode only (no locality or street level geocode). **2** Street level geocode only (no locality or parcel level geocode). **3** Street and parcel level geocodes (no locality geocode). **4** Locality level geocode only (no street or parcel level geocode). **5** Locality and parcel level geocodes (no street level geocode). **6** Locality and street level geocodes (no parcel level geocodes).

columnName	Description
	7 Locality, street and parcel level geocodes.
AUS.GNAF_PID	The G-NAF Persistent Identifier (G-NAF PID) is a 14-character alphanumeric string that uniquely identifies each G-NAF address. The PID is constructed from a combination of the major address fields of the G-NAF Dictionary. An example of a G-NAF PID is: GAVIC411711441
AUS.GNAF_RELIABILITY	A number indicating the geocode precision. Reliability is related to the dictionary used to determine the geocode. Data with geocoded reliability levels 1, 2, and 3 is contained in the GNAF123 Dictionary. This is point (parcel) level geocoded data. Data with geocoded reliability levels 4, 5, and 6 is contained in the GNAF456 Dictionary. This contains non-parcel centroid geocoded data. **1** Geocode accuracy recorded to appropriate surveying standard. For example, this could apply to an address level geocode that was manually geocoded. Geocode resolution is sufficient to place the centroid within address site boundary with a GPS. **2** Geocode accuracy sufficient to place centroid within address site boundary. For example, this could apply to an address level geocode that was automatically calculated as the centroid of the corresponding cadastre parcel. **3** Geocode accuracy sufficient to place centroid near (or possibly within) address site boundary. For example, this could apply to an address level geocode that was automatically calculated by calculating where on the road the address was likely to appear based upon other bounding geocoded addresses. **4** Geocode accuracy sufficient to associate address site with a unique road feature. For example, this could apply to a street level geocode that was automatically calculated by using the road centerline reference data.

columnName	Description
	5 Geocode resolution sufficient to associate address site with a unique locality or neighborhood. For example, this could apply to a locality level geocode that was automatically calculated as the centroid of the locality. **6** Geocode resolution sufficient to associate address site with a unique region. For example, this could apply to a locality level geocode that was derived from topographic feature.
AUS.LEVEL_NUMBER	The number of a floor or level in a multi-story building. For example, **Floor 2** , 17 Jones Street The G-NAF database includes level information for some Australian states. Level information may be associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.LEVEL_TYPE	The label used for a floor of a multi-story building. For example, "Level" or "Floor". In this example, the level type is "Level": Suite 3 **Level 7**, 17 Jones Street In this example, Suite 3 is a unit. The G-NAF database includes level information for some Australian states. Level information may be

columnName	Description
	associated with unit information, but not necessarily. If the G-NAF database contains multiple records with the same level, the level information is returned only if the input address contains unique content (such as a unit number). If the G-NAF dictionary has level information for an address, that information is returned with the matched candidate. The correct level information is returned (when available) even if the input address did not include level information, or if the input had the wrong level information. If the input address has level information but the G-NAF database does not include level information for the matching address, then the input level information is discarded since it is not validated in the G-NAF data.
AUS.MESH_BLOCK_ID	A Meshblock is the smallest geographic unit for which statistical data is collected by the Australian Bureau of Statistics (ABS). Meshblocks usually contain a minimum of 20 to 50 households. This is about one fifth the size of a Collection District (CD). You can use the Meshblock ID to do additional attributions against your own data.

ReverseAPNLookup

ReverseAPNLookup allows you to look up an address using:

- An Assessor's Parcel Number (APN). An APN is an ID number assigned to a piece of land by a county assessor. An APN is unique only within a county.
- A FIPS county code. A Federal Information Processing Standard (FIPS) code is an ID number assigned to a county by the U.S. Federal government.
- A FIPS state code. A FIPS state code is an ID number assigned to each state by the U.S. Federal government.

These three pieces of information, used together, can uniquely identify a specific parcel. You must use all three pieces of information to perform a lookup using ReverseAPNLookup.

Note: ReverseAPNLookup only works for U.S. addresses for which APN data is available. See the coverage map included with the points database for more information.

ReverseAPNLookup is part of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

Input

ReverseAPNLookup takes an APN, FIPS county code, and FIPS state code as input. The following table provides information on the format and layout of the input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 146: Reverse APNLookup Input Data

columnName	Format	Description/Valid Values
APN	String [45]	The assessor's parcel number (APN) for the property you want to look up.
InputKeyValue	String	User-defined data, such as a record ID or source code.
USFIPSCountyNumber	String [5]	The FIPS county code for the county in which the property resides.
USFIPSStateCode	String [2]	The FIPS state code for the state in which the property resides.

Options

The following table lists the options that control ReverseAPNLookup processing.

Table 147: ReverseAPNLookup Options

optionName	Description/Valid Values
Dataset	Specifies the database to use to look up the parcel. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.
LatLongFormat	Specifies the format for returned latitude/longitude. **Decimal** (90.000000-180.000000) Default. **Integer** (90000000-180000000)

optionName	Description/Valid Values
RetrieveElevation	Specifies whether or not ReverseAPNLookup returns the elevation of the address. Elevation is the distance above or below sea level of a given location. The elevation is returned in the **Elevation** output field, which is part of the Latitude/Longitude output group. **Note:** This option requires that you have licensed and installed the Centrus Premium Points database. Elevation data is not available for all addresses. See the coverage map included with the points database.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputVerbose	Specifies whether or not ReverseAPNLookup provides an additional description field as output. This field provides the text equivalent to a field represented by a code. For example, LocationCode returns a code that indicates the accuracy (quality) of the assigned geocode. LocationCode.Description provides the description for the code returned. One of the following: **Y** Include verbose fields. **N** Do not include verbose fields. Default.
OutputRecordType	Specifies optional data to include in the output. Note that ReverseAPNLookup always returns the default data listed in **Default Output** on page 530. The data you select here is returned with the default output data. • **C**—Census • **L**—Latitude/Longitude • **E**—Parsed Elements • **Q**—Qualifiers • **R**—Range • **S**—Segment

optionName	Description/Valid Values
	For a list of fields included in each record type, see **Output** on page 529. If you do not want all of the fields in a group returned, do not select the group, and instead list only those fields you want returned in **OutputFields**.
OutputFields	Indicate the individual output fields you want returned. List fields with a pipe (|) between each field. You can use this field instead of the **OutputRecordType** field to limit the output to the specific fields you want. Default list: AddressLine1|LastLine|Longitude|Latitude|MatchCode|LocationCode

Output

ReverseAPNLookup returns the following categories of data:

- **Census** on page 529
- **Default Output** on page 530
- **Latitude/Longitude** on page 533
- **Parsed Elements** on page 533
- **Qualifiers** on page 535
- **Range** on page 537
- **Segment** on page 538

The following tables list the output fields included in each category.

Note: The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Census

The Census output fields contain census information from the U.S. 2000 Census. To include census data in the output, set **OutputRecordType** = C.

Table 148: Census Output Fields

columnName	Description
BlockSuffix	Single character block suffix for split Census blocks. Returns A or B. Only available in Centrus Enhanced data.
CBSACode	Indicates Core Based Statistical Area (CBSA).

columnName	Description
CBSADivisionCode	Denotes a subdivision of a CBSA.
CBSADivisionName	Describes a subdivision of a CBSA. Only returned when you set the configuration options to return additional descriptions (verbose).
CBSAMetro	Metropolitan Statistical Area. Valid values include: **Y** Metro statistical area **N** Micro statistical area **null** Data unavailable
CBSAName	Describes CBSA. Only returned when you set the configuration options to return additional descriptions (verbose).
CensusBlockID	The ID of the Census Federal Information Processing Standard (FIPS) code.
CensusTract	Six digits extracted from the CensusBlockID.
CSACode	Denotes the code for a geographic entity that consists of 2 or more adjacent CBSAs with employment interchange measures of at least 15.
CSAName	Describes the name for a geographic entity that consists of 2 or more adjacent CBSAs with employment interchange measures of at least 15. Only returned when you set the configuration options to return additional descriptions (verbose).
USCountyName	Name of the county, including the text "County" or "Parish." Only returned when you set the configuration options to return additional descriptions (verbose).
USFIPSStateCountyCode	Five-digit FIPS code for state and county extracted from the CensusBlockID.

Default Output

ReverseAPNLookup always returns the address, geocode, and result indicators.

Table 149: Default Output Fields

columnName	Description
AdditionalInputData	Mailstop, attention line, or deliver instructions as included in the input data. This field is always null. **Note:** ReverseAPNLookup does not process this information. It simply includes the information as entered in the input data.
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
APN	The Assessor's Parcel Number that was specified in the input.
City	Municipality name.
Confidence	Indicates the confidence in the output provided. The range is from 0 (zero) to 100, with 0 being no match and 100 being an exact match.
Country	The name of the country. Since ReverseAPNLookup only works for U.S. locations, this field will always contain **United States of America**.
Distance	The distance, in feet, of the dwelling along the segment.
Elevation	The distance in feet above or below sea level of the parcel.
FirmName	Name of the company.
LastLine	Complete last address line (municipality, state, and postal code).
Latitude	Seven-digit number in degrees and calculated to 4 decimal places (in the format specified).
LocationCode	Indicates the accuracy (quality) of the assigned geocode. For more information, see **Address Location Codes** on page 575.
Longitude	Seven-digit number in degrees and calculated to 4 decimal places (in the format specified).
MatchCode	Indicates the portions of the address that matched to the directory file. For more information, see **Geocoding Match Codes** on page 592.

columnName	Description
PercentGeocode	The percent along the street segment that matches the geocode. For example, if the returned geocode falls 1/3 along the way of the entire street segment, the percent is 33.000. **Note:** This value is always 0.0 for matches to point-level data and intersections.
PostalCode	Nine-digit ZIP Code with or without a hyphen.
PostalCode.AddOn	Four-digit ZIP Code extension.
PostalCode.Base	Five-digit ZIP Code.
ProcessedBy	The feature code for the stage that processed the request. The value is KGL for ReverseAPNLookup.
StateProvince	Two-character state abbreviation.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	Reason for failure: • Internal System Error • No Address Found • Insufficient Input Data
Status.Description	Description of the problem: **Problem + explanation** Returned when Status.Code = Internal System Error. **Geocoding Failed** Returned when Status.code = No Address Found) **No location returned** Returned when Status.code = No Address Found.
StreetDataType	The data set that ReverseAPNLookup attempted to match against.
StreetSide	Indicates the side of the street for the range.

columnName	Description	
	LEFT	Left side of the street.
	RIGHT	Right side of the street.
	BOTH	Both sides of the street.
USFIPSCountyNumber	Three-digit FIPS county code specified in the input.	
USFIPSStateCode	Two-digit FIPS state code specified in the input.	
USUrbanName	USPS® urbanization name. Puerto Rican addresses only.	

Latitude/Longitude

The latitude/longitude output fields contain the geographic coordinates of the location and elevation. To include latitude/longitude output fields in the output, set **OutputRecordType** = L.

Table 150: Latitude/Longitude Output Fields

columnName	Description
Elevation	The distance in feet above or below sea level of the parcel.
Latitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).
Longitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).

Parsed Elements

The Parsed Elements output fields contain standard address information as individual units, such as street suffixes (AVE) and leading directionals (N and SE). To include parsed elements in the output, set **OutputRecordType** = E.

Table 151: Parsed Elements Output Fields

columnName	Description
ApartmentLabel	Apartment designator (such as STE or APT), for example: 123 E Main St. APT 3

columnName	Description
ApartmentLabel2	Secondary apartment designator, for example: 123 E Main St. APT 3, 4th Floor
ApartmentNumber	Apartment number, for example: 123 E Main St. APT 3
ApartmentNumber2	Secondary apartment number, for example: 123 E Main St. APT 3, 4th Floor
City	Municipality name.
CrossStreetLeadingDirectional	Leading directional, for example: 123 E Main St. Apt 3
CrossStreetName[2]	Cross street name, for example: 123 E Main St. Apt 3
CrossStreetSuffix	Cross street suffix, for example: 123 E Main St. Apt 3
CrossStreetTrailingDirectional	Cross street trailing directional, for example: 123 Pennsylvania Ave NW
HouseNumber	Building number, for example: 123 E Main St. Apt 3 **Note:** This is an approximate building number based on the APN, FIPS county code, and FIPS state code provided. This approximate address may not exist or may not accept mail delivery.
LeadingDirectional	Leading directional, for example: 123 E Main St. Apt 3
PrivateMailbox	Private mailbox indicator. Not output for multiline input.
PrivateMailbox.Designator	The type of private mailbox. Possible values include: • Standard • Non-Standard
RRHC	Rural Route/Highway Contract indicator.
StreetName	Street name, for example: 123 E Main St. Apt 3
StreetSuffix	Street suffix, for example: 123 E Main St. Apt 3

[2] ReverseAPNLookup only returns Cross street outputs if you entered an intersection as an address. For example, entering Pearl and 28th, Boulder, CO returns cross street information. Entering 2800 Pearl, Boulder, CO does NOT return cross street information.

columnName	Description
TrailingDirectional	Trailing directional, for example: 123 Pennsylvania Ave NW

Qualifiers

The qualifiers output fields contain qualification information on the match, such as the location code and the match code. To include qualifier output fields in the output, set **OutputRecordType** = Q.

Table 152: Qualifiers Output Fields

columnName	Description
CountryLevel	The category of postal data available. Always returns A in ReverseAPNLookup—Validates, corrects, and provides missing postal code, city name, state/county name, street address elements, and country name.
DatabaseVersion	USPS publish date, in the format Month Year.
EWSMatch	Indicates if ReverseAPNLookup denied a match because of Early Warning System (EWS) data. **Y** EWS denied a match. **null** EWS did not deny a match. For more information on EWS, see **Early Warning System (EWS)** on page 303.
ExpirationDate	Date the database expires, in the format MM/DD/YY.
GeoStanMatchScore	Record matching score (for multimatches only).
Intersection	Indicates if ReverseAPNLookup found a cross-street match. **T** True, a cross-street match was found. **F** False, a cross-street match was not found.
IsAlias	ReverseAPNLookup located a matched record by an index alias. Returns 3 characters. The first is an **N** for normal street match or **A** for alias match (including buildings, aliases, firms, etc.). The next 2 characters are: **01** Basic index (normal address match)

columnName	Description
	02 USPS street name alias index **03** USPS building index **04** USPS firm name index **05** Statewide intersection alias match (when using the Usw.gsi or Use.gsi file) **06** Spatial data street name alias (when using the Us_pw.gsi, Us_pe.gsi, Us_psw.gsi, or Us_pse.gsi file) **07** Alternate index (when using Zip9.gsu, Zip9e.gsu, and Zip9w.gsu) **08** LACS^{Link}
LACSAddress	Indicates if ReverseAPNLookup converted an address due to the Locatable Address Conversion System (LACS). **L** Converted **null** Not converted For more information on LACS, see **Locatable Address Conversion System (LACS)** on page 303.
LocationCode.Description	LocationCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).
MatchCode.Description	MatchCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).
RecordType	Indicates the record type: • GeneralDelivery • HighRise • FirmRecord • Normal • PostOfficeBox • RRHighwayContract
RecordType.Default	Indicates type of match that occurred for the record type HighRise or RRHighwayContract: **Y** Default match **N** Exact match

columnName	Description
	U Not matched
StreetDataCode	Indicates the data used to obtain a match. **0** USPS data in either the Centrus Enhanced, Centrus GDT, or Centrus NAVTEQ database. **1** TIGER data in the Centrus Enhanced database. **2** TomTom data in the Cenrus GDT database. **6** NAVTEQ data in the Centrus NAVTEQ database. **7** TomTom point-level data in the Centrus TomTom Points database. **8** Point-level data from the Centrus Points database. **9** Auxiliary file data

Range

The range output fields contain information on the street range, such as the high and low unit numbers. To include range data fields in the output, set **OutputRecordType** = R.

Table 153: Range Output Fields

columnName	Description
Alternate	USPS code that specifies whether a record is a base or alternate record. **B** Base record. Base records can represent a range of addresses or an individual address, such as a firm record. **A** Alternate record. Alternate records are individual delivery points.
HouseNumberHigh	House number high.
HouseNumberLow	House number low.
HouseNumberParity	Indicates if the house number range contains even or odd numbers. **E** Even **O** Odd

columnName	Description
	B Both
PostalCodeExtensionHigh	4-digit ZIP Code extension high.
PostalCodeExtensionLow	4-digit Zip Code extension low.
UnitNumberHigh	Unit number high.
UnitNumberLow	Unit number low.
UnitNumberParity	Indicates if the unit number range contains even or odd numbers. **E** Even **O** Odd **B** Both

Segment

The segment output fields contain information on the street segment identified by the data provider. To include segment data fields in the output, set **OutputRecordType** = S.

Table 154: Segment Output Fields

columnName	Description
BlockLeft	Provides the Census FIPS Code that indicates the address is on the left side of the street.
BlockRight	Provides the Census FIPS Code that indicates the address is on the right side of the street.
BlockSuffixLeft	Current left Block suffix for Census 2000 Geography. Returns A or B. Only available in Centrus Enhanced data.
BlockSuffixRight	Current right Block suffix for Census 2000 Geography. Returns A or B. Only available in Centrus Enhanced data.
RoadClass	The type of road: **1** The road is a major road.

columnName	Description
	2 The road is a minor road.
PointCode	Unique point ID assigned by the data provider. This field is blank if the matched record is not from point-level data.
SegmentCode	Unique 10-digit segment ID assigned by the street network provider.
SegmentDirection	Indicates the order of numbers on a segment. **F** Forward **R** Reversed **B** Both **U** Undetermined
SegmentHouseNumberHigh	A high range number in the segment.
SegmentHouseNumberLow	A low range number in the segment.
SegmentLength	The length, in feet, of a block segment.
SegmentParity	Indicates which side of the street has odd numbers. **L** Left side of the street **R** Right side of the street **B** Both sides of the street **U** Undetermined

ReverseGeocodeAddressGlobal

ReverseGeocodeAddressGlobal determines the address for a given latitude/longitude point. ReverseGeocodeAddressGlobal can determine addresses in many countries. The countries available to you depends on which country databases you have installed. For example, if you have databases for Canada, Italy, and Australia installed, ReverseGeocodeAddressGlobal would be able to geocode addresses in these countries in a single stage.

Note: ReverseGeocodeAddressGlobal does not support U.S. addresses. To geocode U.S. addresses, use ReverseGeocodeUSLocation. Also, reverse geocoding is not available for the U.K.

Before you can work with ReverseGeocodeAddressGlobal, you must define a global database resource containing a database for one or more countries. Once you create the database resource, a ReverseGeocodeAddressGlobal will become available in the Management Console, Enterprise Designer, and Interactive Driver. For more information on creating a database resource, see the *Spectrum™ Technology Platform User's Guide*.

ReverseGeocodeAddressGlobal is an optional component of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, see **What is the Enterprise Geocoding Module?** on page 294.

Input

ReverseGeocodeAddressGlobal takes longitude and latitude as input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 155: ReverseGeocodeGlobal Input

columnName	Format	Description / Valid Values
Latitude	String	The latitude of the point for which you want address information.
Longitude	String	The longitude of the point for which you want address information.
Country	String	One of the following: • The name of the country in English. • The two-character ISO 3116-1 alpha-2 country code. • The three-character ISO 3116-1 alpha-3 country code.

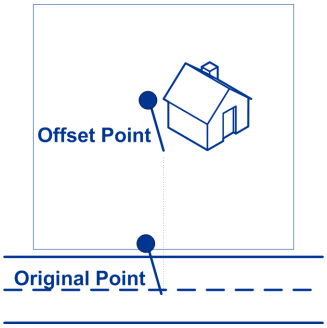
Options

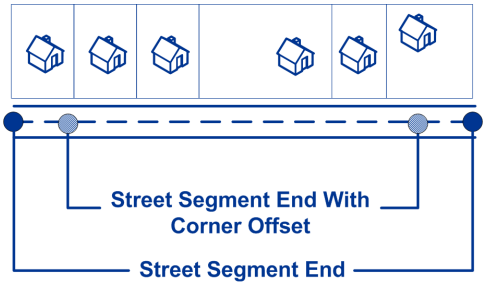
Geocoding Options

Table 156: Geocoding Options

optionName	Description / Valid Values
SearchDistance	The radius from the input coordinates in which to search for an address. Street segments and points within the radius are considered. The default search radius is 150 meters and the maximum search radius is 1600 meters.

optionName	Description / Valid Values
Units	The units in which the search distance is specified. One of the following: • Feet • Miles • Meters • Kilometers
OffsetFromStreet	Indicates the offset distance from the street segments to use in street-level geocoding. The distance is specified in the units you specify in the OffsetUnits option. **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). The default value varies by country: • 10 meters—Australia (AUS), Austria (AUT), Germany (DEU). • 7 meters—Argentina (ARG), Belgium and Luxembourg (BEL), Brazil (BRA), Canada (CAN), Switzerland and Liechtenstein (CHE), Czech Republic (CZE), Chile (CHL), China (CHN), Denmark (DNK), Spain (ESP), Estonia (EST), Finland (FIN), France (FRA), United Kingdom (GBR), Greece (GRC), Hong Kong (HKG), Hungary (HUN), Indonesia (IDN), India (IND), Ireland (IRL), Italy (ITA), Latvia (LVA), Lithuania (LTU), Macau (MAC), Mexico (MEX), Malaysia (MYS), The Netherlands (NLD), Norway (NOR), New Zealand (NZL), Philippines (PHL) Poland (POL), Portugal (PRT), Russia (RUS), Singapore (SGP), Slovakia (SVK), Slovenia (SVN), Sweden (SWE), Thailand (THA), Turkey (TUR), Taiwan (TWN), South Africa (ZAF). The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point.

optionName	Description / Valid Values
	 Street coordinates are accurate to 1/10,000 of a degree and interpolated points are accurate to the millionths of a degree.
OffsetFromCorner	Specifies the distance to offset the street end points in street-level matching. The distance is specified in the units you specify in the OffsetUnits option.This value is used to prevent addresses at street corners from being given the same geocode as the intersection. **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). The default value varies by country: • 12 meters—Australia (AUS), Austria (AUT), Germany (DEU) • 7 meters—Argentina (ARG), Belgium and Luxembourg (BEL), Brazil (BRA), Canada (CAN), Switzerland and Liechtenstein (CHE), Chile (CHL), China (CHN), Czech Republic (CZE), Denmark (DNK), Spain (ESP), Estonia (EST), Finland (FIN), France (FRA), United Kingdom (GBR), Greece (GRC), Hong Kong (HKG), Hungary (HUN), Indonesia (IDN), India (IND), Ireland (IRL), Italy (ITA), Latvia (LVA), Lithuania (LTU), Macau (MAC) Mexico (MEX), Malaysia (MYS), The Netherlands (NLD), Norway (NOR), New Zealand (NZL), Poland (POL), Portugal (PRT), Russia (RUS), Singapore (SGP), Slovakia (SVK), Slovenia (SVN), Sweden (SWE), Thailand (THA), Turkey (TUR), Taiwan (TWN), South Africa (ZAF). The following diagram compares the end points of a street to offset end points.

optionName	Description / Valid Values
	
OffsetUnits	Specifies the unit of measurement for the street offset and corner offset options. One of the following: **Note:** Offset is not supported for the United Kingdom (GBR) or Japan (JPN). • Feet • Miles • Meters • Kilometers The default is Meters.
CoordinateSystem	A coordinate system is a reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry. Spectrum™ Technology Platform supports systems recognized by the European Petroleum Survey Group (EPSG). Each country supports different coordinate systems. Depending on the country, you have one or more of the following options: **EPSG:4230** **EPSG:4283** Also known as the GDA94 coordinate system. **EPSG:4301** Also known as the Tokyo coordinate system. **EPSG:4326** Also known as the WGS84 coordinate system. **EPSG:27200** Also known as the NZGD49 coordinate system. **EPSG:27700** Also known as the British National Grid system.

Matching Options

Table 157: Matching Options

optionName	Description / Valid Values
KeepMultimatch	Specifies whether to return results when the coordinates matche to multiple candidate addresses in the database. If this option is not selected, coordinates that results in multiple address candidates will fail to geocode. If you select this option, specify the maximum number of candidates to return using the MaxCandidates option (see below). One of the following: **Y** Yes, return candidates when multiple candidates are found. Default. **N** No, do not return candidates. Addresses that result in multiple candidates will fail to geocode.
MaxCandidates	If you specify KeepMultimatch=Y, this option specifies the maximum number of results to return. The default is 1.

Data Options

The Data tab allows you to specify which databases to use in reverse geocoding. Databases contain the address and geocode data necessary to determine the address for a given point. The following table lists the options available for specifying the search order of databases.

Table 158: Data Options

optionName	Description / Valid Values
DatabaseSearchOrder	The name of one or more database resources to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*. You can specify multiple database resources. If you specify more than one database, list them in order of preference. The order of the databases has an effect when there are close match candidates from different databases. The close matches that are returned come from the database that is first in the search list. Close matches from lower ranked databases are demoted to non-close matches.

optionName	Description / Valid Values
	You can also use the order of the databases to perform fallback processing if you have an both an address point database and a street-level database installed for the country. List the address point database first and the street database second. If the address cannot be geocoded to the address point level, the geocoder will attempt to geocode it to the street level.

Output

Note: The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section .

Table 159: Reverse Geocode Address Global Output Fields

columnName	Description
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
ApartmentLabel	The type of unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
City	The municipality name. For Japan, the municipality subdivision (oaza).
County	The meaning of county varies by country: • ARG (Argentina)—Department • AUS (Australia)—The Local Government Authority (LGA) • AUT (Austria)—Not used • BEL (Belgium)—Province • BRA (Brazil)—Not used • CAN (Canada)—Not used • CHE (Switzerland)—Province • CHL (Chile)—District • CHN (China)—Not used • COL (Columbia)—TBD • CZE (Czech Republic)—District name or alias

columnName	Description
	• DEU (Germany)—Kreis • DNK (Denmark)—Province • ESP (Spain)—Province • EST (Estonia)—District • FIN (Finland)—Not used • FRA (France)—Department • GBR (Great Britain)—Not used • GRC (Greece) —District • HKG (Hong Kong)—Not used • HUN (Hungary)—District • IDN (Indonesia)—Regency/City • IND (India)—District • IRL (Ireland)—Province • ITA (Italy)—Province • JPN (Japan)—City (shi) • LTU (Lithuania)—County • LVA (Latvia)—District • MAC (Macau)—Not used • MEX (Mexico)—Province • MYS (Malaysia)—Not used • NLD (The Netherlands)—Province • NOR (Norway)—District (fylke/counties) • NZL (New Zealand)—Region • PHL (Philippines)—District • POL (Poland)—District (poviat) • PRT (Portugal)—Not used • RUS (Russia) —District • SGP (Singapore)—Not used • SVK (Slovakia)—Town • SVN (Slovenia)—Region • SWE (Sweden)—Kommun • THA (Thailand)—District (amphoe) • TUR (Turkey)—District • TWN (Taiwan)—Not used • ZAF (South Africa)—District **Note:** If you enter addresses with the Andorran country code (AND), only Andorran addresses are returned, including the AND country code. If you enter an Andorran with a country code of Spain (ESP), you will still get Andorran candidates, but with the ESP country code.

columnName	Description
Distance	The distance from input location in meters. If the input coordinates are an exact match for the address, the value is 0.
FirmName	Name of the company or a place name.
Geocoder.MatchCode	Indicates how closely the input coordinates match the candidate address. For more information, see **Reverse Geocoding Codes (R category)** on page 601.
HouseNumber	The building number for the matched location. For Japan, this field contains the lot number.
HouseNumberHigh	The highest house number of the range in which the address resides.
HouseNumberLow	The lowest house number of the range in which the address resides.
HouseNumberParity	Indicates if the house number range contains even or odd numbers or both. **E** Even **O** Odd **B** Both **U** Unknown
LastLine	Complete last address line (city, state/province, and postal code).
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
Locality	The meaning of locality varies by country: • ARG (Argentina)—Neighborhood or barrio • AUS (Australia)—Not used • AUT (Austria)—Not used • BEL (Belgium)—Not used • BRA (Brazil)—Locality • CAN (Canada)—Dissemination Area and Enumeration Area (DA and EA) • CHE (Switzerland)—Not used • CHL (Chile)—Locality

columnName	Description
	• CHN (China)—Locality • COL (Columbia)—Locality TBD • CZE (Czech Republic)—Locality • DEU (Germany)—Not used • DNK (Denmark)—Not used • ESP (Spain)—Locality • EST (Estonia)—Locality • FIN (Finland)—Not used • FRA (France)—Not used • GBR (Great Britain)—Locality • GRC (Greece) —Locality • HKG (Hong Kong) —Locality • HUN (Hungary)—Locality • IDN (Indonesia)—Village • IND (India)—Locality • IRL (Ireland)—Not used • ITA (Italy)—Locality • JPN (Japan)—City district (chome) • LTU (Lithuania)—Locality • LVA (Latvia)—Locality • MAC (Macau) —Locality • MEX (Mexico)—Locality • MYS (Malaysia)—Not used • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Suburb • PHL (Philippines)—Locality • POL (Poland)—Not used • PRT (Portugal)—Locality • RUS (Russia)—Locality • SGP (Singapore)—Not used • SVK (Slovakia)—Locality • SVN (Slovenia)—Locality • SWE (Sweden)—Not used • THA (Thailand)—Not used • TUR (Turkey)—Locality • TWN (Taiwan)—Locality • ZAF (South Africa)—Locality

columnName	Description
	Geocode Address Global can geocode addresses in which the Locality is input without the City. For example, the following addresses are geocoded identically: calle nicaragua 4705, palermo, capital federal calle nicaragua 4705, palermo
NumberOfCandidateRanges	Indicates the number of ranges of which the candidate is a member. A candidate may be a part of multiple ranges if the candidate is a street instead of a building. To specify the number of ranges to return for each candidate, use the `MaxRanges` option.
NumberOfRangeUnits	Indicates the number of units included in the range. A unit is an address within a building, such as an apartment or office suite. To specify the number of units to return for each range, use the `MaxRangeUnits` option.
PostalCode	The postcode for the address. The format of the postcode varies by country.
PostalCode.Addon	The second part of a postcode. For Canadian addresses this will be the LDU. This field is not used by most countries.
PreAddress	Miscellaneous information that appears before the street name.
PrivateMailbox	This field is not currently used.
SegmentCode	A unique ID that identifies a street segment. In Japan, this is the Jusho code. A Jusho Code is a point ID that represents a unique address.
SegmentParity	Indicates which side of the street has odd numbers. **L** Left side of the street **R** Right side of the street **B** Both sides of the street **U** Undetermined
StateProvince	The meaning of StateProvince varies by country: • ARG (Argentina)—Region or province • AUS (Australia)—State • AUT (Austria)—Region

columnName	Description
	• BEL (Belgium)—Not used • BRA (Brazil)State— • CAN (Canada)—Province • CHE (Switzerland)—Not used • CHL (Chile)—State • CHN (China)—Province • COL (Columbia)—State TBD • CZE (Czech Republic)—Region name or alias. For example, the region HLAVNÍ MESTO PRAHA can be aliased as Prag. • DEU (Germany)—Bundesland • DNK (Denmark)—Not used • ESP (Spain)—Region • EST (Estonia)—Not used • FIN (Finland)—Region (län) • FRA (France)—Region • GBR (Great Britain)—Region • GRC (Greece) —Region • HKG (Hong Kong) —Province • HUN (Hungary)—County • IDN (Indonesia)—Province • IND (India)—State • IRL (Ireland)—Not used • ITA (Italy)—Region • JPN (Japan)—Prefecture • LTU (Lithuania)—Not used • LVA (Latvia)—State • MAC (Macau) — State • MEX (Mexico)—Province • MYS (Malaysia)—State (negeri) • NLD (The Netherlands)—Not used • NOR (Norway)—Not used • NZL (New Zealand)—Region • PHL (Philippines)—State • POL (Poland)—Province (voivodship) • PRT (Portugal)—Region • RUS (Russia) —Region • SGP (Singapore)—Not used • SVK (Slovakia)—Not used • SVN (Slovenia)—Not used • SWE (Sweden)—Region (lan)

columnName	Description
	• THA (Thailand)—Province (changwat) • TUR (Turkey)—Province • TWN (Taiwan)—County or Special Municipality • ZAF (South Africa)—State
StreetDataType	The default search order rank of the database used to geocode the address. A value of "1" indicates that the database is first in the default search order, "2" indicates that the database is second in the default search order, and so on. The default database search order is specified in the Management Console with the Database Resources tool.
StreetName	The street name. In Japan, this contains the block. Japanese addresses typically do not have street names.
StreetPrefix	The type of street when the street type appears before the base street name. For example, AVENUE: 12 **AVENUE** B KALGOORLIE WA 6430
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 3rd St. N.
UnitNumberHigh	The highest unit number of the range in which the unit resides.
UnitNumberLow	The lowest unit number of the range in which the unit resides.

ReverseGeocodeUSLocation

ReverseGeocodeUSLocation takes a latitude and longitude point as input and returns the address that is the best match for that point. For example, you could enter the following information:

Longitude: -105239771 Latitude: 40018912 Search Distance: 150 feet

This input would result in the following output:

4750 WALNUT ST BOULDER, CO 80301-2538
MatchCode = NS0
LocCode = AS0
Lon = -105239773
Lat = 40018911
Distances:
Search = 150
Offset = 50
Squeeze = 50
Nearest = 50.0
Pct Geocode = 94.0
SegID = 472881795
PtID = GDT
Block = 080130122032066
County Name = BOULDER COUNTY
DPBC = 50

Note: The address returned is an approximate address based on the latitude and longitude provided. This approximate address may not exist or may not accept mail delivery.

ReverseGeocodeUSLocation is part of the Enterprise Geocoding Module. For more information on the Enterprise Geocoding Module, including a listing of other components included with it, see **What is the Enterprise Geocoding Module?** on page 294.

ReverseGeocodeUSLocation processes geocodes in the following order:

1. ReverseGeocodeUSLocation defines a small rectangle based on your input geocode and search distance.
2. ReverseGeocodeUSLocation computes the distance between each street segment and the input location.
3. If one segment is closest, ReverseGeocodeUSLocation finds the offset and interpolated percentage (using the squeeze factor) and the street side. It then computes an approximate house number based on this information.

   If there is more than one segment that is equally close to the input location, a multi-match occurs. ReverseGeocodeUSLocation returns the information for all of the equally close segments so that you can determine which segment is applicable.
4. ReverseGeocodeUSLocation returns the address information, including the segment range, the approximate house number, and the parity of the range along with other standard address information.

Note: Although many of the standard address matching outputs apply to the reverse geocoding option, several outputs are unavailable (such as LACS^{Link} information and unit numbers). ReverseGeocodeUSLocation returns these outputs as blank. ReverseGeocodeUSLocation also has outputs specific to reverse geocode processing, such as specific match codes and the distance from the input location to the matched segment.

To use ReverseGeocodeUSLocation, you need additional data files, called GSX files. There is an option to install these files when you install the geocoding database. The GSX files must be installed the GSX subdirectory of the geocoding database. If you install the Centrus Enhanced Points, Centrus Premium Points, or Centrus TeleAtlas Points database, you must recreate the GSX files. For information on using the GSX files, see the *Centrus Utilities Manual*. You can find the manual at **www.g1.com/support** .

Input

ReverseGeocodeUSLocation takes longitude and latitude information as input. The following table provides information on the format and layout of the input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 160: ReverseGeocodeUSLocation Input Data

columnName	Format	Description / Valid Values
Latitude	String	Latitude of the point for which you want address information returned. Specify latitude in millionths of decimal degrees.
Longitude	String	Longitude of the point for which you want address information returned. Specify longitude in millionths of decimal degrees.

Options

ReverseGeocodeUSLocation allows you to set default processing options through the Management Console. You can override certain settings for individual calls to ReverseGeocodeUSLocation using the API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

The following tables provides information on the processing options.

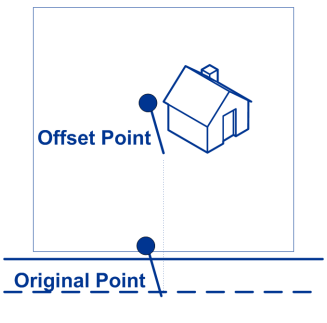
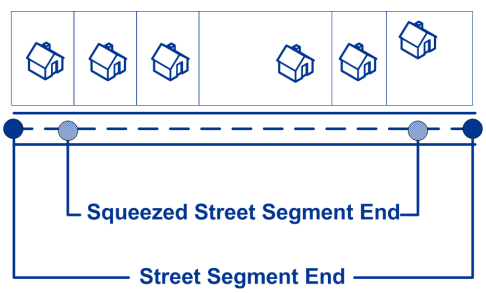
Configuration Options

The following table lists the configuration options for ReverseGeocodeUSLocation.

Table 161: ReverseGeocodeUSLocation Configuration Options

optionName	Description/Valid Values
Dataset	The name of the database that contains the data to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.
SearchDistance	Specifies the radius, in feet, that ReverseGeocodeUSLocation search for matches. The range is 0 - 5280 feet, with a default value of 150 feet.
FindNearestAddress	Specifies whether or not ReverseGeocodeUSLocation should find the nearest address to the input geocode.

optionName	Description/Valid Values
	One of the following: **Y** Find the nearest address. Default. **N** Do not find the nearest address. **Note:** You can use this option with the FindNearestIntersection option to geocode to both addresses and intersections.
FindNearestUnranged	Specifies whether or not ReverseGeocodeUSLocation can match to a street segment that does not have a number range. This option is active when specify FindNearestAddress=Y. One of the following: **Y** Allow ReverGeocodeUSLocation to match to an unranged street segment. Default. **N** Do not allow ReverseGeocodeUSLocation to match to an unranged street segment. **Note:** If you are using the point-level data option, ReverseGeocodeUSLocation ignores the Nearest Unranged option.
FindNearestIntersection	Specifies whether or not ReverseGeocodeUSLocation should find the nearest street intersection to the input geocode. One of the following: **Y** Find the nearest street intersection. Default. **N** Do not find the nearest street intersection. **Note:** You can use this option with the FindNearestAddress option to geocode to both addresses and intersections.
RetrieveAPN	Specifies whether or not ReverseGeocodeUSLocation should determine the address's APN (assessor's parcel number). The APN is an ID number assigned to a property by the local property tax authority. The APN is returned in the **APN** output field, which is part of the Census output group. **Note:** This option requires that you have licensed and installed the Cenrus Enhanced Points or Centrus Premium Points database. APN data is not available for all addresses. See the coverage map included with the points database.

optionName	Description/Valid Values
Offset	Specifies the offset distance from the street segments. The range is 0 - 5280 feet, with the default value of 50 feet. The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point.  Street coordinates are accurate to 1/10,000th of a degree and interpolated points are accurate to 1/1,000,000th of a degree.
Squeeze	Specifies the distance, in feet, to squeeze the street end points in street-level geocoding. The range is 0 -2147483647 feet, with the default value of 50 feet. The following diagram compares the end points of a street to squeezed end points. 
LatLonFormat	Specifies the format to use for returned latitude/longitude.

optionName	Description/Valid Values
	Decimal The format is 90.000000-180.000000. Default. **Integer** The format is 90000000-180000000.
RetrieveElevation	Specifies whether or not ReverseGeocodeUSLocation returns the elevation of the address. Elevation is the distance above or below sea level of a given location. The elevation is returned in the **Elevation** output field, which is part of the Latitude/Longitude output group. **Note:** This option requires that you have licensed and installed the Centrus Premium Points database. Elevation data is not available for all addresses. See the coverage map included with the points database.

Output Format

The following table lists the options that control the format of the output .

Table 162: ReverseGeocodeUSLocation Output Format Options

optionName	Description/Valid Values
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputVerbose	Specifies whether or not ReverseGeocodeUSLocation provides an additional description field as output. This field provides the text equivalent to a field represented by a code. For example, LocationCode returns a code that indicates the accuracy (quality) of the assigned geocode. LocationCode.Description provides the description for the code returned. One of the following: **Y** Include verbose fields.

optionName	Description/Valid Values	
	N	Do not include verbose fields. Default.

Output Data

The following table lists the options that control which data is returned by ReverseGeocodeUSLocation.

Table 163: ReverseGeocodeUSLocation Output Data Options

optionName	Description / Valid Values
OutputRecordType	Specifies the optional data to include in the output. Note that ReverseGeocodeUSLocation always returns the data listed in **Default Output** on page 559. The data you select here is returned with the default output data. • **C**—Census • **L**—Latitude/Longitude • **E**—Parsed Elements • **Q**—Qualifiers • **R**—Range • **S**—Segment For a list of the fields included in each data type, see **Output** on page 557. If you do not want all of the fields in a record type returned, do not specify the record type. Instead, list only those fields you want returned in OutputFields.
OutputFields	Specifies the individual output fields you want returned. List fields with a pipe (|) between each field. You can use this field instead of the Output Record Type to limit the output to those fields that are important to your data needs. Default list: AddressLine1|LastLine|Longitude|Latitude|MatchCode|LocationCode

Output

ReverseGeocodeUSLocation returns the following categories of data:

- **Census** on page 558
- **Default Output** on page 559
- **Latitude/Longitude** on page 561

- **Parsed Elements** on page 562
- **Qualifiers** on page 563
- **Range** on page 565
- **Segment** on page 566

The following tables lists the output fields included in each category.

Note: The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Census

The Census output record type contains census information from the U.S. 2000 Census. To include census data in the output, set **OutputRecordType** = C.

Table 164: Census Output Fields

columnName	Description
APN	The assessor's parcel number of the property. The assessor's parcel number is an ID number assigned to a property by the local property tax authority.
BlockSuffix	Single character block suffix for split Census blocks. Returns A or B. Only available in Centrus Enhanced data.
CBSACode	Indicates Core Based Statistical Area (CBSA).
CBSADivisionCode	Denotes a subdivision of a CBSA.
CBSADivisionName	Describes a subdivision of a CBSA. Only returned when you set the configuration options to return additional descriptions (verbose).
CBSAMetro	Metropolitan Statistical Area. Valid values include: **Y** Metro statistical area. **N** Micro statistical area. **null** Data unavailable.
CBSAName	Describes CBSA. Only returned when you set the configuration options to return additional descriptions (verbose).
CensusBlockID	The ID of the Census Federal Information Processing Standard (FIPS) code.

columnName	Description
CensusTract	6-digits extracted from the CensusBlockID.
CSACode	Denotes the code for a geographic entity that consists of 2 or more adjacent CBSAs with employment interchange measures of at least 15.
CSAName	Describes the name for a geographic entity that consists of 2 or more adjacent CBSAs with employment interchange measures of at least 15. Only returned when you set the configuration options to return additional descriptions (verbose).
USCountyName	Name of the county, including the text "County" or "Parish." Only returned when you set the configuration options to return additional descriptions (verbose).
USFIPSCountyNumber	3-digit FIPS county code extracted from the CensusBlockID.
USFIPSStateCode	2-digit FIPS state code extracted from the CensusBlockID.
USFIPSStateCountyCode	5-digit FIPS code for state and county extracted from the CensusBlockID.

Default Output

ReverseGeocodeUSAddress always includes the following fields in the output.

Table 165: Default Output Fields

columnName	Description
AdditionalInputData	Mailstop, attention line, or deliver instructions as included in the input data. **Note:** ReverseGeocodeUSLocation does not process this information. It simply includes the information as entered in the input data.
AddressLine1	First line of the address.
AddressLine2	Second line of the address.
City	Municipality name.

columnName	Description
Confidence	Indicates the confidence in the output provided. The range is from 0 (zero) to 100, with 0 being no match and 100 being an exact match.
Country	Country name.
Distance	The distance, in feet, of the dwelling along the segment.
Elevation	The location's elevation in feet above or below sea level.
FirmName	Name of the company.
LastLine	Complete last address line (municipality, state, and postal code).
Latitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).
LocationCode	Indicate the accuracy (quality) of the assigned geocode. For more information, see **Address Location Codes** on page 575.
Longitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).
MatchCode	Indicates the portions of the address that matched to the directory file. For more information, see **Geocoding Match Codes** on page 592.
PercentGeocode	The percent along the street segment that matches the geocode. For example, if the returned geocode falls 1/3 along the way of the entire street segment, the percent is 33.000. **Note:** This value is always 0.0 for matches to point-level data and intersections.
PostalCode	9-digit ZIP Code with or without a hyphen.
PostalCode.AddOn	4-digit ZIP Code extension.
PostalCode.Base	5-digit ZIP Code.
ProcessedBy	The underlying software that processed the request. KGR for ReverseGeocodeUSLocation.

columnName	Description
RRHC	Rural Route Highway Contract (RRHC). This field is null if the address not a RRHC.
StateProvince	2-character state abbreviation.
Status	Reports the success or failure of the match attempt **null** Success **F** Failure
Status.Code	Reason for failure: • Internal System Error • No Geocode Found • Insufficient Input Data
Status.Description	Description of the problem: **Problem + explanation** Returned when Status.Code contains "Internal System Error." **Geocoding Failed** Returned when Status.code contains "No Geocode Found". **No location returned** Returned when Status.code contains "No Geocode Found."
StreetDataType	The data set that ReverseGeocodeUSLocation attempted to match against.
StreetSide	Indicates the side of the street for the range. **LEFT** Left side of the street **RIGHT** Right side of the street **BOTH** Both sides of the street
USUrbanName	Urbanization name. Puerto Rico addresses only.

Latitude/Longitude

The latitude/longitude output fields contain the geographic coordinates of the location. To include latitude/longitude output fields in the output, set **OutputRecordType** = L.

Table 166: Latitude/Longitude Output Fields

columnName	Description
Elevation	The location's elevation in feet above or below sea level.
Latitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).
Longitude	7-digit number in degrees and calculated to 4 decimal places (in the format specified).

Parsed Elements

The Parsed Elements output record type contains standard address information as individual units, such as street suffixes (AVE) and leading directionals (N and SE). To include parsed elements in the output, set **OutputRecordType** = E.

Table 167: Parsed Elements Output Fields

columnName	Description
ApartmentLabel	Unit, such as apartment, suite, or lot.
ApartmentLabel2	Unit, such as apartment, suite, or lot.
ApartmentNumber	Unit number.
ApartmentNumber2	Unit number.
CrossStreetLeadingDirectional	Prefix for cross street.
CrossStreetName	Name of cross street.
CrossStreetSuffix	Cross street suffix.
CrossStreetTrailingDirectional	Postfix for cross street.
HouseNumber	Building number for the matched location. **Note:** This is an approximate building number based on the latitude and longitude provided. This approximate address may not exist or may not accept mail delivery.

columnName	Description
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.
PrivateMailbox	Private mailbox. Not output for multiline input.
PrivateMailbox.Designator	Private mailbox description. Not output for multiline input.
StreetName	Street name.
StreetSuffix	The street type of the matched location. For example, AVE for Avenue.
TrailingDirectional	Street directional that follows the street name. For example, the N in 456 Washington N.

Qualifiers

The qualifiers output record type contains qualification information on the match, such as the location code and the match code. To include latitude/longitude output fields in the output, set **OutputRecordType** = Q.

Table 168: Qualifiers Output Fields

columnName	Description
CountryLevel	The category of postal data available. Always returns A in ReverseGeocodeUSLocation—Validates, corrects, and provides missing postal code, city name, state/county name, street address elements, and country name.
DatabaseVersion	USPS publish date, in the format Month Year.
EWSMatch	Indicates if ReverseGeocodeUSLocation denied a match because of Early Warning System (EWS) data. **Y** EWS denied a match. **null** EWS did not deny a match. For more information on EWS, see **Early Warning System (EWS)** on page 303.
ExpirationDate	Date the database expires, in the format MM/DD/YY.

columnName	Description
GeoStanMatchScore	Record matching score (for multimatches only).
Intersection	Indicates if ReverseGeocodeUSLocation found a cross-street match. **T** True, a cross-street match was found. **F** False, a cross-street match was not found.
IsAlias	ReverseGeocodeUSLocation located a matched record by an index alias. Returns 3 characters. The first is an **N** for normal street match or **A** for alias match (including buildings, aliases, firms, etc.). The next 2 characters are: **01** Basic index (normal address match) **02** USPS street name alias index **03** USPS building index **04** USPS firm name index **05** Statewide intersection alias match (when using the Usw.gsi or Use.gsi file) **06** Spatial data street name alias (when using, the Us_pw.gsi, Us_pe.gsi, Us_psw.gsi, or Us_pse.gsi file is required) **07** Alternate index (when using Zip9.gsu, Zip9e.gsu, and Zip9w.gsu) **08** LACS^{Link}
LACSAddress	Indicates if ReverseGeocodeUSLocation converted an address due to the Locatable Address Conversion System (LACS). **L** Converted **null** Not converted. For more information on LACS, see **Locatable Address Conversion System (LACS)** on page 303.
LocationCode.Description	LocationCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).
MatchCode.Description	MatchCode converted to text. Only returned when you set the configuration options to return additional descriptions (verbose).

columnName	Description
RecordType	Indicates the record type: • GeneralDelivery • HighRise • FirmRecord • Normal • PostOfficeBox • RRHighwayContract
RecordType.Default	Indicates type of match that occurred for the record type HighRise or RRHighwayContract: **Y** Default match. **N** Exact match. **U** Not matched.
StreetDataCode	Indicates the data used to obtain a match. **0** USPS data in either the Centrus Enhanced, Centrus GDT, or Centrus NAVTEQ database. **1** TIGER data in the Centrus Enhanced database. **2** TeleAtlas data in the Centrus GDT database. **6** NAVTEQ data in the Centrus NAVTEQ database. **7** TeleAtlas point-level data in the Centrus TeleAtlas Points database. **8** Point-level data from the Centrus Points database. **9** Auxiliary file data.
StreetDataType	Indicates the data first used to attempt a match.

Range

The range output record type contains information on the street range, such as the high and low unit numbers. To include range data fields in the output, set **OutputRecordType** = R.

Table 169: Range Output Fields

columnName	Description
Alternate	USPS code that specifies whether a record is a base or alternate record. **B** Base record. Base records can represent a range of addresses or an individual address, such as a firm record. **A** Alternate record. Alternate records are individual delivery points.
HouseNumberHigh	House number high.
HouseNumberLow	House number low.
HouseNumberParity	Indicates if the house number range contains even or odd numbers. **E** Even **O** Odd **B** Both
PostalCodeExtensionHigh	4-digit ZIP Code extension high.
PostalCodeExtensionLow	4-digit Zip Code extension low.
UnitNumberHigh	Unit number high.
UnitNumberLow	Unit number low.
UnitNumberParity	Indicates if the unit number range contains even or odd numbers. **E** Even **O** Odd **B** Both

Segment

The segment output record type contains information on the street segment identified by the data provider. To include segment data fields in the output, set **OutputRecordType** = S.

Table 170: Segment Output Fields

columnName	Description
BlockLeft	Provides the Census FIPS Code that indicates the address is on the left side of the street.
BlockRight	Provides the Census FIPS Code that indicates the address is on the right side of the street.
BlockSuffixLeft	Current left Block suffix for Census 2000 Geography. Returns A or B. Only available in Centrus Enhanced data.
BlockSuffixRight	Current right Block suffix for Census 2000 Geography. Returns A or B. Only available in Centrus Enhanced data.
RoadClass	The type of road: **1** Major **2** Minor
SegmentCode	Unique 10-digit segment ID assigned by the street network provider.
SegmentDirection	Indicates the order of numbers on a segment. **F** Forward **R** Reversed **B** Both **U** Undetermined
SegmentHouseNumberHigh	A high range number in the segment.
SegmentHouseNumberLow	A low range number in the segment.
SegmentLength	The length, in feet, of a block segment.
SegmentParity	Indicates which side of the street has odd numbers. **L** Left side of the street **R** Right side of the street **B** Both sides of the street

columnName	Description	
	U	Undetermined

Geocode US Address Auxiliary Files

Auxiliary File Overview

Use auxiliary files to match against special data that is not included in the GeocodeUSAddress database.

The GeocodeUSAddress database is updated regularly to incorporate changes made by the USPS and third-party data vendors. You may have newer information that has not yet been incorporated. Auxiliary files provide a way for you to process your input records against a file that includes these changes.

Note: ReverseGeocodeUSAddress does not support auxiliary files.

There are two types of auxiliary file records:

- **Street Records**—Contains a range of one or more addresses on a street. For required fields, see **Auxiliary File Layout** on page 570. A street record must not have secondary address information mailstops, Private mail boxes (PMBs), and PO Boxes.
- **Landmark Records**—Represents a single site. For required fields, see **Auxiliary File Layout** on page 570. A landmark record must not have street type abbreviations, predirectional and postdirectional abbreviations, or low and high house numbers.

Note: You cannot update the auxiliary file while GeocodeUSAddress is running. If you want to update the auxiliary file, stop GeocodeUSAddress before attempting to replace or edit the file.

Matching to Auxiliary Files

GeocodeUSAddress matches an input address to an auxiliary file as follows:

1. GeocodeUSAddress determines if there is an auxiliary file present.

   If you have an auxiliary file in the dataset directory, GeocodeUSAddress automatically loads and attempts to match to the auxiliary file. You can verify that GeocodeUSAddress found an auxiliary file by looking at the version information page in the Management Console. One of the following statuses display:

   - Loaded—An auxiliary file is loaded
   - None—An auxiliary file has not been found or loaded
   - Invalid—An auxiliary file was found, but failed to successfully load

   GeocodeUSAddress only accepts one auxiliary file. If more than one auxiliary files is present, GeocodeUSAddress attempts to match against the first file. GeocodeUSAddress ignores any additional auxiliary files for matching, regardless if GeocodeUSAddress found a match to the first auxiliary file.

If a record in the auxiliary files is invalid, GeocodeUSAddress returns a invalid record message. GeocodeUSAddress continues to match input addresses with the auxiliary file, but will not match to the invalid auxiliary file record.

2. If an auxiliary file is present, GeocodeUSAddress attempts to match to the auxiliary file.

   GeocodeUSAddress assumes that the auxiliary file is the most accurate data set and attempts to find a match to the input address in the auxiliary file. If GeocodeUSAddress cannot find a match in the auxiliary file, it matches the input address with the other Enterprise Geocoding Module databases.

   Note: GeocodeUSAddress only matches input address lists to auxiliary files if there is an exact match. Your input address list should be free of misspellings and incomplete addresses.

3. If GeocodeUSAddress finds an exact record match to the auxiliary file, it standardizes the match to USPS regulations and returns the output of the auxiliary file match.

   GeocodeUSAddress uses the following defaults if you do not include the values in the auxiliary file:

   - House number parity = B (both odds and evens)
   - Segment direction = A (ascending)
   - Side of street = U (unknown)

Record Type Matching Rules

When matching against an auxiliary file, GeocodeUSAddress uses the following rules:

Street record match

- The input house number must fall within or be equal to the low and high house number values of the auxiliary record.
- The input house number must agree with the parity of the auxiliary record.
- The input ZIP Code must exactly match the ZIP Code of the auxiliary record.

Landmark record match

- The input data must contain a ZIP Code and address line, and the values must exactly match the values on the auxiliary record.
- The input address cannot have any other data, such as a house number, unit number, or Private Mail Box (PMB).

Note: GeocodeUSAddress only matches the ZIP Code against the auxiliary file. GeocodeUSAddress does not verify that the ZIP Code of the input address record is correct for the city and state. Validate this information in your input lists before processing against the auxiliary file.

Unavailable Features and Functions

The following features and functions do not apply when GeocodeUSAddress makes an auxiliary file match.

- GeocodeUSAddress does not match to
- two-line addresses
- multi-line addresses
- intersection addresses
- dual addresses

- GeocodeUSAddress does not perform EWS, ZIPMove, LACSLink, or DPV processing on auxiliary matches
- You can only access the auxiliary file with processing through the Find function. You cannot access the auxiliary file through the Find First/Next or MBR functions
- You can only access the auxiliary file logic using the address code option of the Find function, not the geocode option.

Auxiliary Match Output

GeocodeUSAddress provides special data type, match codes, and location code values for auxiliary matches. When GeocodeUSAddress finds a match to an auxiliary file, the default output follows these conventions:

- GeocodeUSAddress formats the auxiliary file match as a street-style address for output. This excludes PO Boxes, Rural Routes, General Delivery, etc.
- GeocodeUSAddress follows the case setting you indicate (by default, upper case) by the casing function. GeocodeUSAddress does not maintain the casing in the auxiliary file for mixed casing values. For example, GeocodeUSAddress returns O'Donnell as ODONNELL or Odonnell depending on the setting of the casing function.

Note: GeocodeUSAddress does not change the casing for the User Data field.

- GeocodeUSAddress removes spaces at the beginning and ending of fields in the auxiliary file.

Note: GeocodeUSAddress does not remove spaces for the User Data field.

Auxiliary File Layout

You must comply with the following organizational rules when you create auxiliary file:

- Files are fixed-width text files with a `.gax` extension
- Files can contain up to 500,000 records.
- Use semicolons in the first column to indicate a row is a comment, not a data record; GeocodeUSAddress ignores rows that begin with a semicolon.
- For optimal performance, order the records within the file by descending ZIP Code, and then descending street name.
- Records must represent only one side of a street. To represent both sides of a street, create a record for each side of the street.
- Records must represent segments that are straight lines.
- House numbers must follow USPS rules documented in Publication 28.
- Numeric fields, such as ZIP Codes, must contain only numbers.
- If house numbers are present in the record, the house number range must be valid according to USPS rules documented in Publication 28, Appendix E.
- Latitude and Longitude values must be in millionths of decimal degrees.
- Records cannot contain PO Box addresses.

The following table shows auxiliary file layout.

Table 171: Auxiliary File Layout

Field	Description	Required	Required for Street Segment Match	Required for Landmark Match	Exact Match Required if Present	Length	Position
ZIP Code	5-digit ZIP Code.	X	X	X	X	5	1-5
Street name	Name of the street or landmark.	X	X	X	X	30	6-35
Street type abbreviation	Street type. Also called street suffix. See the USPS Publication 28, Appendix C for a complete list of supported street types.				X	4	36-39
Predirectional	USPS street name predirectional abbreviation. Supported values are N, E, S, W, NE, NW, SE, and SW.				X	2	40-41
Postdirectional	USPS street name postdirectional abbreviations. Supported values are N, E, S, W, NE, NW, SE, and SW.				X	2	42-43
RESERVED	RESERVED					4	44-47
Low house number	Low house number of the address range.	X	X			11	48-58
High house number	High house number of the address range.	X	X			11	59-69

Field	Description	Required	Required for Street Segment Match	Required for Landmark Match	Exact Match Required if Present	Length	Position
House number parity	Indicates the parity of the house number in the range. E - Even O - Odd B - Both					1	70
Segment direction	Direction the house numbers progress along the segment: F - Forward (default) R - Reverse					1	71
RESERVED	RESERVED					1	72
FIPS state	US government FIPS state code.					2	73-74
FIPS county	US government FIPS county code.					3	75-77
Census tract	US Census tract number.					6	78-83
Census block group	US Census block group number.					1	84
Census block ID	US Census block ID number.					3	85-87
RESERVED	RESERVED					5	88-92
State abbreviation	USPS state abbreviation.					2	93-94
County name	Name of the county.					25	95-119

Field	Description	Required	Required for Street Segment Match	Required for Landmark Match	Exact Match Required if Present	Length	Position
MCD code	Minor Civil Division code.					5	120-124
MCD name	Minor Civil Division name.					40	125-164
CBSA code	Core Based Statistical Area code.					5	165-169
CBSA name	Core Based Statistical Area name.					49	170-218
RESERVED	RESERVED					5	219-223
City Name	City name. Overrides the city/state preferred city name upon a return.					40	224-263
RESERVED	RESERVED					237	264-500
User-defined data	User-defined data.					300	501-800
Record ID Number	User-defined unique record identifier.					10	801-810
Side of street	Side of the street for the address: L - Left side R - Right side B - Both sides U - Unknown side (default) This is relative to the segment endpoints					1	811

Field	Description	Required	Required for Street Segment Match	Required for Landmark Match	Exact Match Required if Present	Length	Position
	and the segment direction.						
Beginning longitude	Beginning longitude of the street segment in millionths of degrees.	X	X	X		11	812-822
Beginning latitude	Beginning latitude of the street segment in millionths of degrees.	X	X	X		10	823-832
Ending longitude	Ending longitude of the street segment in millionths of degrees.					11	833-843
Ending latitude	Ending latitude of the street segment in millionths of degrees.					10	844-853

Location Codes for U.S. Geocoding

Geocoding components return location codes indicating the accuracy of the assigned geocode. There are three types of geocodes:

- **Address location codes**—Location codes that begin with an "A" are address location codes. Address location codes indicate a geocode made directly to a street network segment (or two segments, in the case of an intersection). For more information, see **Address Location Codes** on page 575.
- **Street centroid location codes**—Location codes that begin with "C" are street centroid location codes. Street centroids may be returned if the street centroid fallback option is enabled and an address-level geocode could not be determined. For more information about street centroid location codes, see **Street Centroid Location Codes** on page 585.
- **ZIP + 4 centroid location codes**—Location codes that begin with a "Z" are ZIP + 4 centroid location codes. ZIP + 4 centroids indicate a geocode could not be determined for the address, so the location of the center of the address's ZIP + 4 was returned instead. For more information, see **ZIP + 4 Centroid Location Codes** on page 586.
- **Geographic centroid location codes**—Location codes that begin with "G" are geographic centroid location codes. Geographic centroids may be returned if the street centroid fallback option is enabled

and an address-level geocode could not be determined. For more information about street centroid location codes, see **Street Centroid Location Codes** on page 585.

- **Location unavailable**—Location codes that begin with "E" indicate that neither an address location nor a ZIP + 4 centroid could be determined. This usually occurs when you have requested ZIP Code centroids of a high quality, and one is not available for that match. For more information, see **Address Location Codes** on page 575.

Address Location Codes

Address location codes detail the known qualities about the geocode.

An address location code has the following characters.

1st character	Always an A indicating an address location.	
2nd character	May be one of the following	
	C	Interpolated address point location
	G	Auxiliary file data location
	I	Application infers the correct segment from the candidate records
	P	Point-level data location
	R	Location represents a ranged address
	S	Location on a street range
	X	Location on an intersection of two streets
3rd and 4th character	Digit indicating other qualities about the location.	

Table 172: Address Location Codes

Code		Description
AGn		Indicates an auxiliary file for a geocode match where n is one of the following values:
	n = 0	The geocode represents the center of a parcel or building.
	n = 1	The geocode is an interpolated address along a segment.
	n = 2	The geocode is an interpolated address along a segment, and the side of the street cannot be determined from the data provided in the auxiliary file record.
	n = 3	The geocode is the midpoint of the street segment.
APnn		Indicates a point-level geocode match representing the center of a parcel or building, where nn is one of the following values:
	nn = 02	Parcel centroid Indicates the center of an accessor's parcel (tract or lot) polygon. When the center of an irregularly shaped parcel falls outside of its polygon, the centroid is manually repositioned to fall inside the polygon as closely as possible to the actual center.
	nn = 04	Address points Represents field-collected GPS points with field-collected address data.

Code	Description
nn = 05	Structure centroid Indicates the center of a building footprint polygon, where the building receives mail or has telephone service. Usually a residential address consists of a single building. For houses with outbuildings (detached garages, shed, barns, etc.), only the residences have a structure point. Condominiums and duplexes have multiple points for each building. Larger buildings, such as apartment complexes, typically receive mail at one address for each building and therefore individual apartments are not represented as discrete structure points. Shopping malls, industrial complexes, and academic or medical center campuses where one building accepts mail for the entire complex are represented as one point. When addresses are assigned to multiple buildings within one complex, each addressed structure is represented by a point. If the center of a structure falls outside of its polygon, the center is manually repositioned to fall inside the polygon.
nn = 07	Manually placed Address points are manually placed to coincide with the midpoint of a parcel's street frontage at a distance from the center line.
nn = 08	Front door point

Code		Description
		Represents the designated primary entrance to a building. If a building has multiple entrances and there is no designated primary entrance or the primary entrance cannot readily be determined, the primary entrance is chosen based on proximity to the main access street and availability of parking.
	nn = 09	Driveway offset point Represents a point located on the primary access road (most commonly a driveway) at a perpendicular distance of between 33-98 feet (10-30 meters) from the main roadway.
	nn = 10	Street access point Represents the primary point of access from the street network. This address point type is located where the driveway or other access road intersects the main roadway.
	nn=21	Base parcel point When unable to match to an input unit number, or when the unit number is missing from an address location with multiple units, the "base" parcel information is returned, the address is not standardized to a unit number, and additional information, such as an Assessor's Parcel Number, is not returned.
AIn		The correct segment is inferred from the candidate records at match time.

Code	Description
ASn	House range address geocode. This is the most accurate geocode available.
AIn and ASn share the same qualities for n as follows:	
n = 0	Best location.
n = 1	Street side is unknown. The Census FIPS Block ID is assigned from the left side; however, there is no assigned offset and the point is placed directly on the street.
n = 2	Indicates one or both of the following: • The address is interpolated onto a TIGER segment that did not initially contain address ranges. • The original segment name changed to match the USPS spelling. This specifically refers to street type, predirectional, and postdirectional. **Note:** Only the second case is valid for non-TIGER data because segment range interpolation is only completed for TIGER data.
n = 3	Both 1 and 2.
n = 7	Placeholder. Used when starting and ending points of segments contain the same value and shape data is not available.
ACnh	

Code		Description
	The ACnn 4th digit characteristics are as follows:	
	n = 0	Represents the interpolation between two points, both coming from User Dictionaries.
	n = 1	Represents the interpolation between two points. The low boundary came from a User Dictionary and the high boundary, from a non-User Dictionary.
	n = 2	Represents the interpolation between one point and one street segment end point, both coming from User Dictionaries.
	n = 3	Represents the interpolation between one point (low boundary) and one street segment end point (high boundary). The low boundary came from a User Dictionary and the high boundary from a non-User Dictionary.
	n = 4	Represents the interpolation between two points. The low boundary came from a non-User Dictionary and the high boundary from a User Dictionary.
	n = 5	Represents the interpolation between two points, both coming from non-User Dictionaries.
	n = 6	Represents the interpolation between one point (low boundary) and one street segment end point (high boundary). The low boundary came from a non-User Dictionary and the high boundary from a User Dictionary.

Code	Description
n = 7	Represents the interpolation between one point and one street segment end point and both came from non-User Dictionaries.
n = 8	Represents the interpolation between one street segment end point andone point, both coming from User Dictionaries.
n = 9	Represents the interpolation between one street segment end point (low boundary) andone point (high boundary). The low boundary came from a User Dictionary and the high boundary from a non-User Dictionary.
n = A	Represents the interpolation between two street segment end points, both coming from User Dictionaries.
n = B	Represents the interpolation between two street segment end points. The low boundary came from a User Dictionary and the high boundary from a non-User Dictionary.
n = C	Represents the interpolation between one street segment end point (low boundary) and one point (high boundary). The low boundary came from a non-User Dictionary and the high boundary from a User Dictionary.
n = D	Represents the interpolation between one street segment end point and one point, both coming from non-User Dictionary.

Code		Description
	n = E	Represents the interpolation between two street segment end points. The low boundary came from a non-User Dictionary and the high boundary from a User Dictionary.
	n = F	Represents the interpolation between two street segment end points, both coming from non-User Dictionaries.
ARn		Ranged address geocode, where n is one of the following:
	n = 1	The geocode is placed along a single street segment, midway between the interpolated location of the first and second input house numbers in the range.
	n = 2	The geocode is placed along a single street segment, midway between the interpolated location of the first and second input house numbers in the range, and the side of the street is unknown. The Census FIPS Block ID is assigned from the left side; however, there is no assigned offset and the point is placed directly on the street.
	n = 4	The input range spans multiple USPS segments. The geocode is placed on the endpoint of the segment which corresponds to the first input house number, closest to the end nearest the second input house number.
	n = 7	Placeholder. Used when the starting and ending points of the matched segment contain the

Code		Description
		same value and shape data is not available.
AXn		Intersection geocode, where n is one of the following:
	n = 3	Standard single-point intersection computed from the center lines of street segments.
	n = 8	Interpolated (divided-road) intersection geocode. Attempts to return a centroid for the intersection.

Table 173: Match Codes for No Match

Code		Description
Ennn		Indicates an error, or no match. This can occur when the address entered does not exist in the database, or the address is badly formed and cannot be parsed correctly. The last three digits of an error code indicate which parts of an address the application could not match to the database.
	nnn = 000	No match made.
	nnn = 001	Low level error.
	nnn = 002	Could not find data file.
	nnn = 003	Incorrect GSD file signature or version ID.
	nnn = 004	GSD file out of date. Only occurs in CASS mode.

Code	Description
nnn = 010	No city and state or ZIP Code found.
nnn = 011	Input ZIP not in the directory.
nnn = 012	Input city not in the directory.
nnn = 013	Input city not unique in the directory.
nnn = 014	Out of licensed area. Only occurs if using Pitney Bowes Software licensing technology.
nnn = 015	Record count is depleted and license has expired.
nnn = 020	No matching streets found in directory.
nnn = 021	No matching cross streets for an intersection match.
nnn = 022	No matching segments.
nnn = 023	Unresolved match.
nnn = 024	No matching segments. (Same as 022.)
nnn = 025	Too many possible cross streets for intersection matching.
nnn = 026	No address found when attempting a multiline match.
nnn = 027	Invalid directional attempted.
nnn = 028	Record also matched EWS data, therefore the application denied the match.

Code	Description
nnn = 029	No matching range, single street segment found.
nnn = 030	No matching range, multiple street segments found.

Street Centroid Location Codes

Street centroid location codes indicate the Census ID accuracy and the position of the geocode on the returned street segment. A street centroid location code has the following characters.

1st character	Always C indicating a location derived from a street segment.
2nd character	Census ID accuracy based on the search area used to obtain matching Street Segment.
3rd character	Location of geocode on the returned street segment.

The following table contains the values and descriptions for the location codes.

Character position	Code	Description
2nd Character		
	B	Block Group accuracy (most accurate). Based on input ZIP Code.
	T	Census Tract accuracy. Based on input ZIP Code.
	C	Unclassified Census accuracy. Normally accurate to at least the County level. Based on input ZIP Code.
	F	Unknown Census accuracy. Based on Finance area.

Character position	Code	Description
3rd Character	P	Unknown Census accuracy. Based on input City.
	C	Segment Centroid.
	L	Segment low-range end point.
	H	Segment high-range end point.

ZIP + 4 Centroid Location Codes

ZIP + 4 centroid location codes indicate the quality of two location attributes: Census ID accuracy and positional accuracy.

A ZIP + 4 centroid location code has the following characters.

1st character	Always Z indicating a location derived from a ZIP centroid.
2nd character	Census ID accuracy.
3rd character	Location type.
4th character	How the location and Census ID was defined. Provided for completeness, but may not be useful for most applications.

Table 174: ZIP + 4 Centroid Location Codes

Character Position	Code	Description
2nd Character		
	B	Block Group accuracy (most accurate).
	T	Census Tract accuracy.

Character Position	Code	Description
	C	Unclassified Census accuracy. Normally accurate to at least the County level.
3rd Character		
	5	Location of the Post Office that delivers mail to the address, a 5-digit ZIP Code centroid, or a location based upon locale (city). See the 4th character for a precise indication of locational accuracy.
	7	Location based upon a ZIP + 2 centroid. These locations can represent a multiple block area in urban locations, or a slightly larger area in rural settings.
	9	Location based upon a ZIP + 4 centroid. These are the most accurate centroids and normally place the location on the correct block face. For a small number of records, the location may be the middle of the entire street on which the ZIP + 4 falls. See the 4th character for a precise indication of locational accuracy.
4th Character		
	A	Address matched to a single segment. Location assigned in the middle of the matched street segment, offset to the proper side of the street.
	a	Address matched to a single segment, but the correct side of the street is unknown. Location assigned in the middle of the

Character Position	Code	Description
		matched street segment, offset to the left side of the street, as address ranges increase.
	B	Address matched to multiple segments, all segments have the same Block Group. Location assigned to the middle of the matched street segment with the most house number ranges within this ZIP + 4. Location offset to the proper side of the street.
	b	Same as methodology B except the correct side of the street is unknown. Location assigned in the middle of the matched street segment, offset to the left side of the street, as address ranges increase.
	C	Address matched to multiple segments, with all segments having the same Census Tract. Returns the Block Group representing the most households in this ZIP + 4. Location assigned to t he middle of the matched street segment with the most house number ranges within this ZIP + 4. Location offset to the proper side of the street.
	c	Same as methodology C except the correct side of the street is unknown. Location assigned in the middle of the matched street segment, offset to the left side of the street, as address ranges increase.
	D	Address matched to multiple segments, with all segments having the same County. Returns

Character Position	Code	Description
		the Block Group representing the most households in this ZIP + 4. Location assigned to the middle of the matched street segment with the most house number ranges within this ZIP + 4. Location offset to the proper side of the street.
	d	Same as methodology D except the correct side of the street is unknown. Location assigned in the middle of the matched street segment, offset to the left side of the street, as address ranges increase.
	E	Street name matched; no house ranges available. All matched segments have the same Block Group. Location placed on the segment closest to the center of the matched segments. In most cases, this is on the mid-point of the entire street.
	F	Street name matched; no house ranges available. All matched segments have the same Census Tract. Location placed on the segment closest to the center of the matched segments. In most cases, this is on the mid-point of the entire street.
	G	Street name matched (no house ranges available). All matched segments have the same County. Location placed on the segment closest to the center of the matched segments. In most cases, this is on the mid-point of the entire street.

Character Position	Code	Description
	H	Same as methodology G, but some segments are not in the same County. Used for less than .05% of the centroids.
	I	Created ZIP + 2 cluster centroid as defined by methodologies A, a, B, and b. All centroids in this ZIP + 2 cluster have the same Block Group. Location assigned to the ZIP + 2 centroid.
	J	Created ZIP + 2 cluster centroid as defined by methodologies A, a, B, b, C, and c. All centroids in this ZIP + 2 cluster have the same Census Tract. Location assigned to the ZIP + 2 centroid.
	K	Created ZIP + 2 cluster centroid as defined by methodologies A, a, B, b, C, c, D, and d. Location assigned to the ZIP + 2 centroid.
	L	Created ZIP + 2 cluster centroid as defined by methodology E. All centroids in this ZIP + 2 cluster have the same Block Group. Location assigned to the ZIP + 2 centroid.
	M	Created ZIP+2 cluster centroid as defined by methodology E and F. All centroids in this ZIP + 2 cluster have the same Census Tract. Location assigned to the ZIP + 2 centroid.
	N	Created ZIP + 2 cluster centroid as defined by methodology E, F, G, and H. Location assigned to the ZIP + 2 centroid.

Character Position	Code	Description
	V	Over 95% of addresses in this ZIP Code are in a single Census Tract. Location assigned to the ZIP Code centroid.
	W	Over 80% of addresses in this ZIP Code are in a single Census Tract. Reasonable Census Tract accuracy. Location assigned to the ZIP Code centroid.
	X	Less than 80% of addresses in this ZIP Code are in a single Census Tract. Census ID is uncertain. Location assigned to the ZIP Code centroid.
	Y	Rural or sparsely populated area. Census code is uncertain. Location based upon the USGS places file.
	Z	P.O. Box or General Delivery addresses. Census code is uncertain. Location based upon the Post Office location that delivers the mail to that address.

Geographic Centroid Location Codes

Geographic centroid location codes indicate the quality a city, county, or state centroid. A geographic centroid location code has the following characters.

1st character	Always G indicating a location derived from a geographic centroid.
2nd character	Geographic area type. One of the following: **M** Municipality (for example, a city) **C** County **S** State

Match Codes for U.S. Geocoding

Geocoding Match Codes

Geocoding components return match codes indicating the address portions that matched or did not match to the database. If the geocoder cannot make a match, the match code begins with E and the remaining digits indicate why the address did not match. The digits do not specifically refer to which address elements did not match, but rather why the address did not match.

Table 175: Match Codes

Code	Description
Ahh	Same as Shh, but indicates match to an alias name record or an alternate record.
Chh	The street address did not match, but the geocoder located a street segment based on the input ZIP Code or city .
D00	Matched to a small town with P.O. Box or General Delivery only.
Gxx	Matched to an auxiliary file.
Hhh	The house number was changed.
Jhh	Matched to a user-defined dictionary.
Nxx	Matched to the nearest address. Used with reverse geocoding. The following are the only values for N: **NSO** Nearest street center match (nearest street segment interpolated) **NS1** Nearest unranged street segment **NP0** Nearest point address **NX0** Nearest intersection
P	Successful reverse APN lookup.

Code	Description
Qhh	Matched to USPS range records with unique ZIP Codes. CASS rules prohibit altering an input ZIP if it matches a unique ZIP Code value.
Rhh	Matched to a ranged address.
Shh	Matched to USPS data. This is considered the best address match, because it matched directly against the USPS list of addresses. S is returned for a small number of addresses when the matched address has a blank ZIP + 4.
Thh	Matched to a street segment record. Street segment records do not contain ZIP Code information. If you enter a ZIP Code, the application returns the ZIP Code you entered. If the input city and state has only one ZIP Code, the application returns that ZIP Code.
Uhh	Matched to USPS data but cannot resolve the ZIP + 4 code without the firm name or other information. CASS mode returns an E023 (multiple match) error code.
Xhhh	Matched to an intersection of two streets, for example, "Clay St & Michigan Ave." The first hex digit refers to the last line information, the second hex digit refers to the first street in the intersection, and the third hex digit refers to the second street in the intersection. **Note:** The USPS does not allow intersections as a valid deliverable address.
Yhhh	Same as Xhhh, but an alias name record was used for one or both streets.
Z	No address given, but verified the provided ZIP Code .

The following table contains the description of the hex digits for the match code values.

Table 176: Description of Hex Digits

Code	In first hex position means:	In second and third hex position means:
0	No change in last line.	No change in address line.
1	ZIP Code changed.	Street type changed.
2	City changed.	Predirectional changed.
3	City and ZIP Code changed.	Street type and predirectional changed.
4	State changed.	Postdirectional changed.
5	State and ZIP Code changed.	Street type and postdirectional changed.
6	State and City changed.	Predirectional and postdirectional changed.
7	State, City, and ZIP Code changed.	Street type, predirectional, and postdirectional changed.
8	ZIP + 4 changed.	Street name changed.
9	ZIP and ZIP + 4 changed.	Street name and street type changed.
A	City and ZIP + 4 changed.	Street name and predirectional changed.
B	City, ZIP, and ZIP + 4 changed.	Street name, street type, and predirectional changed.
C	State and ZIP + 4 changed.	Street name and postdirectional changed.
D	State, ZIP, and ZIP + 4 changed.	Street name, street type, and postdirectional changed.
E	State, City, and ZIP + 4 changed.	Street name, predirectional, and postdirectional changed.

Code	In first hex position means:	In second and third hex position means:
F	State, City, ZIP, and ZIP + 4 changed.	Street name, street type, predirectional, and postdirectional changed.

If neither an address location nor a ZIP + 4 centroid can be determined, the location code will start with "E". This occurs infrequently when the component does not have a 5-digit centroid location. Enterprise Geocoding Module components can also return an E location code type when it cannot standardize an input address and there is no input ZIP Code. In this case, do not assume the ZIP Code returned with the non-standardized address is the correct ZIP Code because the component did not standardize the address; therefore, the component does not return geocoding or Census Block information.

Table 177: Match Codes for No Match

Code	Description
Ennn	Indicates an error, or no match. This can occur when the address entered does not exist in the database, or the address is badly formed and cannot be parsed correctly. The last three digits of an error code indicate which parts of an address the application could not match to the database.
nnn = 000	No match made.
nnn = 001	Low level error.
nnn = 002	Could not find data file.
nnn = 003	Incorrect GSD file signature or version ID.
nnn = 004	GSD file out of date. Only occurs in CASS mode.
nnn = 010	No city and state or ZIP Code found.
nnn = 011	Input ZIP not in the directory.

Code	Description
nnn = 012	Input city not in the directory.
nnn = 013	Input city not unique in the directory.
nnn = 014	Out of licensed area. Only occurs if using Pitney Bowes Software licensing technology.
nnn = 015	Record count is depleted and license has expired.
nnn = 020	No matching streets found in directory.
nnn = 021	No matching cross streets for an intersection match.
nnn = 022	No matching segments.
nnn = 023	Unresolved match.
nnn = 024	No matching segments. (Same as 022.)
nnn = 025	Too many possible cross streets for intersection matching.
nnn = 026	No address found when attempting a multiline match.
nnn = 027	Invalid directional attempted.
nnn = 028	Record also matched EWS data, therefore the application denied the match.
nnn = 029	No matching range, single street segment found.
nnn = 030	No matching range, multiple street segments found.

Result Codes for International Geocoding

Introduction

Geocoders return a result code in the Geocoder.MatchCode output field for each attempted record match.

Single Close Match (S category)

Matches in the S category indicate that the record was matched to a single address candidate. The first character (S) indicates that the geocoding server found a street address that matches the record. The second position in the code indicates the positional accuracy of the resulting point for the geocoded record.

Different types of matches:

- **S0**—Single close match with no coordinates available (rare occurrence).
- **S1**—Single close match with the point located at postal code centroid. For Canada, this is an FSA centroid. For Japan, S1 indicates that the candidate matched to prefecture, but not to anything more precise.
- **S3**—Single close match with the point located at postal code centroid. For Canada, this is an FSALDU centroid. For Japan, S3 indicates that the candidate matched to prefecture, city, and municipality subdivision (oaza).
- **S4**—Single close match with the point located at the street centroid. The S4 code is followed by letters and dashes indicating match precision. For information about these letters, see **Match Detail Codes** on page 598.
- **S4G**—For Australia, the S4G result code is used for single matches with a G-NAF Reliability level of 4 (associated with a unique road feature.) The reliability level is returned in the output field AUS.GNAF_Reliability. For more information, see **Australia G-NAF Database Output** on page 420.
- **S5**—Single close match with the point located at a street address position. For Japan, S5 indicates that the candidate matched block but the lot was either not matched or not provided on input. The S5 code is followed by letters and dashes indicating match precision. For information about these letters, see **Match Detail Codes** on page 598.
- **S6**—Single close match with the point located at centroid of geometry postal code. (For example, large buildings having their own codes.)
- **S7**—Single match with the point located at an interpolated point along the candidate's street segment. When the potential candidate is not an address point candidate and there are no exact house number matches among other address point candidates, the S7 result is returned using address point interpolation. The point is interpolated according to the next highest or lowest address point candidate that both intersects the segment and whose house number is contained within the range of houses of the original candidate. By using known address reference points on the street segment, the S7 point can be adjusted to a more accurate position.

  For Australia, the S7-------G result code is also used for single matches with G-NAF Reliability level of 3. The reliability level is returned in the output field AUS.GNAF_Reliability. For more information, see **Australia G-NAF Database Output** on page 420.

- **S8**—Single close match with the point located at either the single point associated with an address point candidate or at an address point candidate that shares the same house number. No interpolation is required.

  For Australia, the S8-------G result code is also used for single matches with G-NAF Reliability levels of 1or 2 (the highest level of G-NAF Reliability. The reliability level is returned in the output field AUS.GNAF_Reliability.

- **SG**—For Australia, single close match with the point at the center of a locality, or Locality level geocode derived from topographic feature. An SG-------G result code is associated with G-NAF Reliability Level 5 (locality or neighborhood) or with Level 6 (unique region). The reliability level is returned in the output field AUS.GNAF_Reliability. For more information, see **Australia G-NAF Database Output** on page 420.
- **SP**—For Australian locations only, single close match to a postal (PO Box) location. This can be generated from the Street Range Address database only (not the G-NAF database).
- **SL** —Sublocality. The SL result code is used for for India only. This indicates a sublocality (block or sector) match when a street match (S4) or point match (S8) was not possible. An SL result code also requires a match on other geographic input fields (city, district, or state).
- **SX**—Single close match with the point located at street intersection.

Match Detail Codes

S4 and S5 geocode results are followed by additional characters that indicate details of the match precision. A typical result code looks like this:

```
S5-P--S--A
```

This code indicates that the geocoder returned a single close match to a street address (S5), with an exact match on street directional prefix (P), street suffix (S), and the match was obtained using a database provided by Pitney Bowes Software (A). It did not match the house number (H), street name (N), street type (T), city (C) or postal code (Z), so these positions contain a dash.

A perfect street level match would return a result code with every component of the address matched and would look like this:

```
S5HPNTSCZA
```

Note: To minimize the match inaccuracy, specify stricter matching conditions.

The match detail codes appear in the order listed in the following table.

Table 178: Matching Details Codes

Character	Description
H	Exact match on house number.
P	Exact match on street directional prefix.
N	Exact match on street name.
T	Exact match on street type.
S	Exact match on street suffix.

Character	Description
C	Exact match on city name.
Z	Exact match on postal code.
A	Address matched using an Enterprise Geocoding Module database provided by Pitney Bowes Software.
U	Address matched using a user-defined geocoding database.
-	If any of the fields do not match, the letter is replaced with a dash.

Best Match From Multiple Candidates (M category)

Matches in the M category indicate that there is more than one close match candidate for the record and Spectrum™ Technology Platform has chosen the best one of those candidates. As in the S category, the second position in the code of M category matches the positional accuracy of the resulting point object.

- **M1**—Multiple close matches, point located at postal code centroid.
- **M3**—Multiple close matches, point located at geographic centroid.
- **M4**—Multiple close matches, point located at the center of a shape point path (shape points define the shape of the street polyline).
- **M5**—Multiple close matches, point located at a street address position (highest accuracy available).
- **M6**—Multiple close matches, point located at point postal code location.
- **MX**—Multiple close matches, point located at street intersection.
- **M0**—Multiple close matches, no coordinates available.

Interpreting S and M Result Codes

For either S or M category result codes, eight additional characters describe how closely the address matches an address in the database. The characters appear in the order listed in the following table. Any non-matched components are represented by a dash.

For example, the result code S5--N-SCZA represents a single close match that matched the street name, street suffix direction, town, and postcode. The dashes indicate that there was no match on house number, street prefix direction, or thoroughfare type. The match came from the Street Range Address database. This record would be geocoded at the street address position of the match candidate.

Category	Description	Example
H	House number	18

Category	Description	Example
P	Street prefix direction P is present if any of these conditions are satisfied: • The candidate pre-directional matches the input pre-directional. • The candidate post-directional matches the input pre-directional after pre- and post-directionals are swapped. • The input does not have a pre-directional.	North
N	Street name	Merivale
T	Street type	St
S	Street suffix direction S in result code is present if any of these conditions are satisfied: • The candidate post-directional matches the input post-directional. • The candidate pre-directional matches the input post-directional after pre- and post-directionals are swapped. • The input does not have a post-directional.	W
C	City name	South Brisbane
Z	Postal code	4101
L	Sublocality. The SL result code is used for for India only. This indicates a sublocality (block or sector) match when a street match (S4) or point match (S8) was not possible. An SL result code also requires a match on other geographic input fields (city, district, or state). In India, street name information is often not available, but a sublocality can be a very small area. Therefore an SL match can be more precise than a street centroid S4 match.	Block-9 Sector-3

Category	Description	Example
A, G, or U	Database type used to obtain the match. • A—Street Range Address database. • G—G-NAF Point Address Dictionary (Australia only). • U—Customer (user-defined) database.	A

Reverse Geocoding Codes (R category)

Matches in the R category indicate that the record was matched by reverse geocoding. The second two characters of the R result code indicate the type of match found. R geocode results include an additional letter to indicate the dictionary from which the match was made.

Example reverse geocoding codes:

Example Reverse Geocoding Code	Description
RS8A	Point/parcel level precision for reverse geocoding. Candidate returned from Address Dictionary.
RS5A	Interpolated street candidate for reverse geocoding. Candidate returned from Address Dictionary.
RS4A	Street centroid candidate for reverse geocoding. Candidate returned from Address Dictionary.
RSLA	Candidate returned from India address dictionary with a sublocality (block or sector) match. In India, street name information is often not available, but a sublocality can be a very small area. Therefore an RSLA match can be more precise than a street centroid RS4 match.

Postal Code Centroid Match (Z category)

Matches in the Z category indicate that no street match was made for one of the following reasons:

- You specified to match to postal code centroids. The resulting point is located at the postal code centroid with four possible accuracy levels.
- There is no close match and you specified to fall back to postal code centroid

The Z category contains the following accuracy levels:

- **Z0**—Postal Code match, no coordinates available (rare occurrence).

- **Z1**—Postal Code centroid match.
- **Z3**—Full postal code centroid match. For Canada, this is an FSALDU centroid.
- **Z6**—Postal Code centroid match for point ZIP.

Geographic Centroid Match (G category)

Matches in the G category indicate that the candidate is located at the geographic centroid with the following possible accuracy levels. Not all levels of accuracy are possible for all countries.

- **G0**—Country centroid. This is not returned for GeocodeAddressWorld.
- **G1**—State or province centroid. For Japan, this indicates a prefecture (ken) match.
- **G2**—County centroid. For Japan, this indicates a city (shi) match.
- **G3**—City centroid. For Japan, this indicates a municipality subdivisio (oaza) match. For Australia, Local Government Authority (LGA) information can be returned from the Street Range Address Database only (not the G-NAF database).
- **G4**—Locality centroid. For Japan, this indicates a city district (chome) match.

Non-match Codes

The following result codes indicate no match was made:

- **N**—No close match.
- **NX**—No close match for street intersections.
- **ND**—Spectrum™ Technology Platform could not find the geocoding database for the given postal code or municipality/state/province.

Custom Databases for International Geocoding

What Is a Custom Database?

A custom database is a table of addresses and coordinates that you can use for geocoding in place of, or in addition to, the standard Enterprise Geocoding Module databases. Use a custom database when you have newer or more precise data than what is available in the standard databases to get more accurate geocoding results. For example, if you have address point data for your customer sites, you can create a custom database of these sites.

A custom database can be used by itself to geocode records, or can be used in combination with the standard databases.

Note: The more user dictionaries you use, the more geocoding performance degrades.

The capabilities of custom databases are as follows.

- All columns supported by normal street geocoding can be included in custom databases.
- Points Of Interest (POI) geocoding is supported in custom databases. Postal or geographic centroid geocoding are not supported in custom databases.
- Custom databases support address browsing using partial street names or Points of Interest.

- A standard database is required to create a custom database. This is because standard databases have some internal structure that must be available when you are creating a custom database.

Requirements

The requirements for custom databases consist of:

- **Source Data Requirements** on page 603
- **Column Requirements** on page 603
- **File Name and Format Requirements** on page 606

Source Data Requirements

Source data for custom databases includes street data, but can also include place names and intersections. Source data for custom databases must conform to the following requirements:

- You can only create custom databases for areas you have licensed. For example, if you create a custom database of Adelaide streets and addresses, you must have a license for South Australia or the entire Australia database.
- Source records must include required columns, which are mapped during the custom database creation process. If a value of a required column is empty for a particular record, that record is not imported into the custom database. Required columns vary by country.
- Source records must be in a MapInfo table (TAB file).
- The MapInfo table (TAB file) must contain specific columns. The database creation utility uses the TAB file to convert the table into the database format. These input columns are described in **Column Requirements** on page 603.
- Intersection segments must have one or more endpoints for Spectrum™ Technology Platform to recognize it as an intersection.
- Source records can be point objects or segments.
- Each row in the table is equal to a street segment.

Column Requirements

You must specify the column names in the MapInfo table (TAB file) for the table to be translated into a custom database. Required columns must be present in the TAB file. Other columns are optional, but recommended. Omitting optional columns may cause negative consequences. If any required columns are missing, an error code is returned.

Australia

Table 179: Required Input Columns for Australia on page 604 lists the required fields.

Table 179: Required Input Columns for Australia

Field	Description	Maximum Column Length	Required
Street Name	Name of street	50	X
Town / Suburb	Municipality	40	X
Postcode	The four-digit postcode.	4	X
State	Abbreviation of state or territory	3	X
Place Name	Place name	40	
Local Government Authority	The LGA.	50	
Unit Number	Unit Numbers are associated with Unit Types,		
Unit Type	Unit Types include Apartment, Building, Flat, Suite, and others.		
Left Starting Address Number	Start of address range on left side of street. For more information, see **Range Order in Custom Databases** on page 608.	11	
Right Starting Address Number	Start of address range on right side of street. For more information, see **Range Order in Custom Databases** on page 608.	11	
Left Ending Address Number	End of address range on left side of street. For more information, see **Range Order in Custom Databases** on page 608.	11	
Right Ending Address Number	End of address range on right side of street. For more information, see **Range Order in Custom Databases** on page 608.	11	

Field	Description	Maximum Column Length	Required
Left Odd/Even indicator	Left side of the street contains only odd or even address ranges (O=odd, E=even, B=both). For more information, see **Odd/Even Indicators in Custom Databases** on page 608. It is strongly recommended that you include this column.	1	
Right Odd/Even indicator	Right side of the street contains only odd or even address ranges (O=odd, E=even, B=both). For more information, see **Odd/Even Indicators in Custom Databases** on page 608. It is strongly recommended that you include this column.	1	

Canada

Table 180: Required Input Columns for Australia on page 605 lists the required fields.

Table 180: Required Input Columns for Australia

Field	Description	Maximum Column Length	Required
Left Start Address	Start of the address range on the left side of the street.	10	X
Right Start Address	Start of the address range on the right side of the street.	10	X
Left End Address	End of the address range on the left side of the street.	10	X
Right End Address	End of the address range on the right side of the street.	10	X
Street Name	Name of street	50	X
Province Abbreviation	Abbreviation of the province	2	X

Field	Description	Maximum Column Length	Required
Left FSA (segment)	FSA for the left side of the street.	3	X
Right FSA (segment)	FSA for the right side of the street.	3	X
Left Odd/Even indicator	Left side of the street contains only odd or even address ranges (O=odd, E=even, B=both). For more information, see **Odd/Even Indicators in Custom Databases** on page 608. It is strongly recommended that you include this column.	1	
Right Odd/Even indicator	Right side of the street contains only odd or even address ranges (O=odd, E=even, B=both). For more information, see **Odd/Even Indicators in Custom Databases** on page 608. It is strongly recommended that you include this column.	1	
City	The city name	28	
Left LDU	LDU for the left side of the street.	3	
Right LDU	LDU for the right side of the street	3	

File Name and Format Requirements

A custom database must meet these requirements:

- The file name must be eight characters or less.
- Each custom database must reside in its own directory.
- The maximum length of a path to a custom database is 1,024 characters.
- The maximum length of the paths to multiple custom databases is 1,024 characters.

Because each custom database resides in its own directory, custom databases may share the same name. However, it is good practice to use a unique name for each custom database.

Some database files are linked to the base name. The other files have fixed names regardless of database. For example, the files for a database called "ud1" would be:

```
postinfo.jdr
```

```
postinfo.jdx

lastline.jdr

post2sac.mmj

geo2sac.mmj

sac2fn_ud.mmj

ud1.jdr

ud1.jdx

ud1.bdx
```

If your data includes place names, the database contains the following files:

```
ud1.pdx

ud1.pbx
```

The database also contains these log files:

```
ud1.log

ud1.err
```

Creating a Custom Database

1. Download the User Dictionary Utility. Go to **www.pbinsight.com/support/product-downloads**, under Applications, click **MapMarker**, then click **MapInfo User Dictionary Utility**.

   Note: The utility uses the term "user dictionary" instead of "custom database". These terms are synonymous.

2. Create a MapInfo tab file (.tab) following the requirements described in **Requirements** on page 603).
3. Start the User Dictionary Utility.
4. Enter the required information and then click **Next**.
   - In the **Country** field, select the country for this custom database.
   - In the **Address Dictionary File** field, accept or browse to the standard database for the selected country. In the **Table Name** field, browse to the TAB file that contains the source data for your custom database.
5. Select the required columns and then click **Next**.

   Note: Field labels and details vary by country. See **Column Requirements** on page 603 for more information.

6. Select optional columns as needed and then click **Next**.
7. Enter the custom database output directory and file name and then click **Finish**.

   You must specify an empty directory. For example, you could browse to C:\data and create an empty directory named TroyUD. Name the file and save it in this directory. For example, the file TroyUD.jdr.

   A log file summarizes the processing and lists all created files. Any errors are indicated in the Error File area of the dialog box.

8. Click **Exit**.

   The custom database directory contains the custom database files (with .mmj, .jdr, .jdx, .sdx, and .bdx file extensions). The log file and error file are also stored in the same directory.

9. Start Management Console and define a new database resource for the custom database. For more information, see the *Spectrum™ Technology Platform User's Guide*. After you define the database resource in the Management Console, select the database in the **Data** tab of the geocoding component.

Range Order in Custom Databases

Enterprise Geocoding Module determines address range order based on a comparison of the start and end addresses. The comparison produces the following results:

- If the end is greater than the start, the range is ascending.
- If the start is greater than the end, the range is descending.
- If the start is equal to the end, the range is ascending.

Odd/Even Indicators in Custom Databases

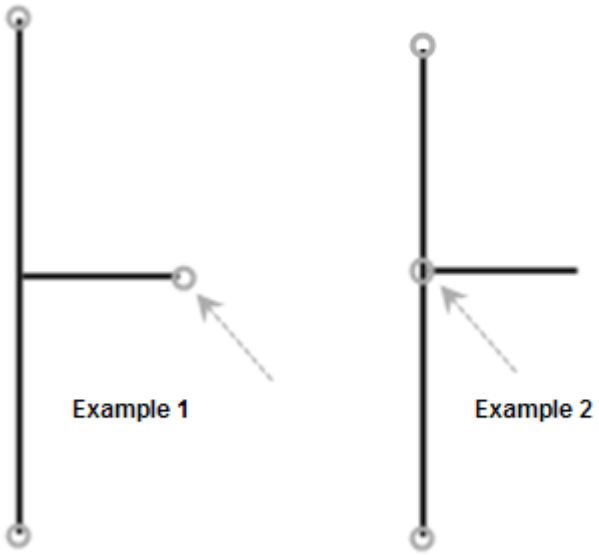
Left and Right Odd/Even Indicator columns specify whether the sides of the street segment contain odd or even address ranges. If your TAB file contains odd/even indicator information, you should use the Odd/Even indicator columns. These columns ensure that geocoded addresses are located on the correct side of the street. Omitting these columns may produce incorrect results. If the Odd/Even Indicator is not included, the following logic is used:

- When the Odd/Even Indicator is **not** specified and both Start Address and End Address have values, the indicator is set to Both, unless the start and end address numbers are the same number. In that case, the indicator is set to Odd if the address numbers are odd, and set to Even if the address numbers are even.
- When the Odd/Even Indicator is **not** specified and both Start Address and End Address have values, the indicator is set to Both (odd and even).

When the Odd/Even Indicator is specified, but is inconsistent with address numbers, the indicator is set to Both.

Street Intersections in Custom Databases

When geocoding street intersections with a custom database, Enterprise Geocoding Module cannot recognize intersections if one or more of the intersection segments does not have an endpoint at the intersection. This can happen when you create a custom database from a customized street table in which some segments that terminate at intersections do not have endpoints (Example"1).



Example 1: Intersection in a custom database does not have endpoints for all segments. Enterprise Geocoding Module does not identify the geocode of this intersection.

Example 2: Intersection in a standard database includes endpoints for all segments. Enterprise Geocoding Module identifies the geocode of this intersection.

Encountering False Positives

What is a False-Positive?

To prevent the generation of address lists, the DPV and LACS^{Link} databases include false-positive records. False-positive records are artificially manufactured addresses that reside in a false-positive table. For each negative response that occurs in a DPV or LACS^{Link} query, a query is made to the false-positive table. A match to this table (called a false-positive match) disables your DPV or LACS^{Link} key. In batch processing the job that contains the violation will complete successfully but you will not be able to run any subsequent jobs that use DPV or LACS^{Link} until you report the violation and obtain a key to reactivate DPV or LACS^{Link}.

Note: The term "seed record violation" is also used to refer to encountering false positive records. The two terms mean the same thing.

Reporting DPV False-Positive Violations

Spectrum™ Technology Platform indicates a false-positive match via messages in the server log.

Client/server calls throw an exception if a false-positive match occurs. When a DPV fals positive record violation occurs, the server log will say:

```
WARN [Log] Seed record violation for S<ZIP, ZIP+4, Address, Unit> ERROR [Log]
 Feature Disabled: DPU: DPV Seed Record Violation. Seed Code: S<Address, ZIP,
 ZIP+4, Unit>
```

Note: If a DPV false positive record is found, the process() method (COM, C++, Java, and .NET) will throw an exception that the feature DPU has been disabled. In C, the processMessage() function will return a non-zero value.

You can report the violation and obtain a restart key by following these steps.

1. In your browser, go to http://<yourserver>:<port>/<product code>/dpv.jsp. For example, http://localhost:8080/unc/dpv.jsp for the Universal Addressing Module and http://localhost:8080/geostan/dpv.jsp for the Enterprise Geocoding Module.
2. Enter the mailer's information into each field. The number in parentheses after each field name indicates the maximum length of the field.
3. Click **Submit** when you're done. A **File Download** dialog will appear.
4. Click **Save** to save the file to your computer. A **Save As** dialog will appear.
5. Specify a file name and location on your local hard drive (for example `c:\DPVSeedFile.txt`) and click **Save**.
6. Go to **www.g1.com/support** and log in.
7. Click **DPV & LACS**[Link] **False Positive**.
8. Follow the on-screen instructions to attach your seed file and obtain a restart key.

DPV False Positive Header File Layout

The USPS® has determined the required layout of the DPV false-positive header file, which is currently defined as a fixed-length file containing two or more 180-byte records. The first record must always be the header record, whose layout is shown below.

Table 181: DPV False-Positive Header Record Layout

Position	Length	Description	Format
1-40	40	Mailer's company name	Alphanumeric
41-98	58	Mailer's address line	Alphanumeric
99-126	28	Mailer's city name	Alphanumeric
127-128	2	Mailer's state abbreviation	Alphabetic
129-137	9	Mailer's 9-digit ZIP Code	Numeric
138-146	9	Total Records Processed	Numeric

Position	Length	Description	Format
147-155	9	Total Records DPV Matched	Numeric
156-164	9	Percent Match Rate to DSF	Numeric
165-173	9	Percent Match Rate to ZIP + 4®	Numeric
174-178	5	Number of ZIP Codes on file	Numeric
179-180	2	Number of False-Positives	Numeric

The trailer record contains information regarding the DPV false-positive match. There must be one trailer record added to the false-positive file for every DPV false-positive match. The layout is shown below.

Table 182: DPV False-Positive Trailer Record Layout

Position	Length	Description	Format
1-2	2	Street predirectional	Alphanumeric
3-30	28	Street name	Alphanumeric
31-34	4	Street suffix abbreviation	Alphanumeric
35-36	2	Street postdirectional	Alphanumeric
37-46	10	Address primary number	Alphanumeric
47-50	4	Address secondary abbreviation	Alphanumeric
51-58	8	Address secondary number	Numeric
59-63	5	Matched ZIP Code	Numeric
64-67	4	Matched ZIP + 4®	Numeric
68-180	113	Filler	Spaces

Reporting LACS/Link False-Positive Violations

Spectrum™ Technology Platform indicates a false-positive match via messages in the server log. Batch jobs will fail if a false-positive match occurs and client/server calls will throw an exception.

Note: The term "seed record violation" is also used to refer to encountering false positive records. The two terms mean the same thing.

When a false positive record is encountered, the server log will say:

```
2005-05-06 17:05:38,978 WARN [com.g1.component.ValidateAddress] Seed record
 violation for RR 2 28562  31373
2005-05-06 17:05:38,978 ERROR [com.g1.component.ValidateAddress] Feature
Disabled: LLU: LACS Seed Record Violation. Seed Code: 28562  31373
2005-05-06 17:05:38,978 ERROR [com.g1.dcg.gateway.Gateway] Gateway exception:
 com.g1.dcg.stage.StageException: com.g1.dcg.component.ComponentException:
Feature Disabled: LLU
2005-05-06 17:06:30,291 ERROR [com.g1.dcg.server.license.policy.Policy]
Feature LACSLink Real- time is disabled.
```

Note: If a LACS^{Link} false positive record is found, the process() method (COM, C++, Java, and .NET) will throw an exception that the feature LLU has been disabled. In C, the processMessage() function will return a non-zero value.

1. In your browser, go to http://<ServerName>:<port>/<product code>/lacslink.jsp. For example, http://localhost:8080/unc/lacslink.jsp for the Universal Addressing Module and http://localhost:8080/geostan/lacslink.jsp for the Enterprise Geocoding Module.
2. Enter the mailer's information into each field. The number in parentheses after the field name indicates the maximum length of the field. Click **Submit** when you're done. A **File Download** dialog will appear.
3. Click **Save** to save the file to your computer. A **Save As** dialog will appear.
4. Specify a file name and location on your local hard drive (for example `c:\lacslink.txt`) and click **Save**.
5. Go to **www.g1.com/support** and log in.
6. Click **DPV & LACS^{Link} False Positive**.
7. Follow the on-screen instructions to attach your seed file and obtain a restart key.

13

Enterprise Routing Module

In this section:

- What Is the Enterprise Routing Module? 614
- GetTravelBoundary 614
- GetTravelCostMatrix 624
- GetTravelDirections 634
- Routing Database Resources 645

What Is the Enterprise Routing Module?

The Enterprise Routing Module provides the ability to obtain driving or walking directions, calculate drive time and drive distance, and identify locations within a certain time or distance from a starting point.

Enterprise Routing Components

- **Get Travel Boundary**—Allows you to obtain polygons corresponding to an isochrone or isodistance calculation. An isochrone is a polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time. An isodistance is a polygon or set of points representing the area that is a certain distance from the starting point.
- **Get Travel Cost Matrix**—Calculates the travel time and distances between an array of start and end locations.
- **Get Travel Directions**—Allows you to route from one point to another point or from one point to multiple other points.

GetTravelBoundary

GetTravelBoundary determines a drive or walk time or distance boundary from a location. This feature obtains polygons corresponding to an isochrone or isodistance calculation. An isochrone is a polygon or set of points representing an area that can be traversed in a network from a starting point in a given amount of time. An isodistance is a polygon or set of points representing the area that is a certain distance from the starting point. It takes a starting latitude and longitude point and an ending latitude and longitude point as input and returns the route that is either fastest or shortest, depending on how you configure the stage.

GetTravelBoundary is part of the Enterprise Routing Module. For more information on this module, including a listing of other components included with it, see **Enterprise Routing Components** on page 614.

To familiarize yourself with how GetTravelBoundary works, use the Management Console's Preview tab to step through the following procedure.

1. Open the Management Console.
2. Under the Services node, select `Get Travel Boundary`.
3. On the **Options** tab, specify the options you want. For information on the options, see **Travel Options** on page 618.
4. Click the **Preview** tab.
5. Enter latitude information into the Latitude cell and longitude information into the Longitude cell. Alternatively, you can paste this information into the cell, even in multiples. If you have a two-column set of data separated by tabs, you can highlight the data and paste it into the first Latitude cell. It will populate as many rows (of both latitude and longitude) as your data requires.
6. Click **Run Preview**.
7. Under Preview Output, you will see the IsoNodeResponse and the IsoPolygonResponse.

Input

GetTravelBoundary takes cost, cost unit, point latitude, and point longitude as input. The following table provides information on the format and layout of the input.

Note: If you intend to interact with the GetTravelBoundary service, note that it is only available as a web service. The Get Travel Cost Matrix is not available through the Java, C++, C, .NET, or COM APIs.

GetTravelBoundary input is formatted according to the structure defined in the Get Travel Boundary WSDL. You can access the WSDL on your Spectrum™ Technology Platform server at:

http://<servername>:<port>/services/GetTravelBoundary?wsdl

Table 183: GetTravelBoundary Input Data

columnName	Format	Description / Valid Values
Latitude	String	Latitude of the point. Specify latitude in the format chosen in the CoordinateFormat option.
Longitude	String	Longitude of the point. Specify longitude in the format chosen in the CoordinateFormat option.
TravelBoundaryCost	String	(Optional) The cost distance or time, in the units specified by either the TravelBoundaryCostUnits field or the DefaultTravelBoundaryCostUnits option. For example, if the unit specified is miles and you specify 10 in this field, the cost would be 10 miles. Use this field to override the default travel boundary cost on a record-by-record basis.
TravelBoundaryCostUnits	String	(Optional) The type of metric used to calculate the travel boundary. One of the following: • Feet • Hours • Kilometers • Meters • Miles • Minutes • Seconds • Yards Use this field to override the default travel boundary cost units on a record-by-record basis.

Options

GetTravelBoundary allows you to set default processing options through the Management Console graphical user interface. You can override certain settings for individual calls to GetTravelBoundary using the Client API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

If you do not specify system-level defaults through the Management Console or a custom interface, and GetTravelBoundary does not pass any processing options at the time of the call, Spectrum™ Technology Platform uses the default options listed in the following sections.

The following table lists the configuration options for GetTravelBoundary.

Input Options

Table 184: GetTravelBoundary Input Options

optionName	Description/Valid Values
DataSetResourceName	The name of the database that contains the data to use in the search process. Use the database name specified in the Enterprise Routing Module Routing Database Resource tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.
CoordinateSystem	The coordinate system of the input coordinates. The format must be the European Petroleum Survey Group (EPSG) code, the SRID code, or the MAPINFO code. Specify the coordinate reference system in the format `codespace:code`. For example, a geometry in Latitude/Longitude (WGS 84) would be `EPSG:4326`. The default value is EPSG:4326, which is also known as Latitude/Longitude (WGS 84). This is a worldwide system and is the terrestrial reference frame associated with the NAVSTAR Global Positioning System (GPS) used extensively for navigation and surveying. This datum is earth-centered ("geocentric") and defined with satellite and terrestrial data. The following table lists some of the more widely used EPSG codes: **Coordinate System** / **EPSG Code** Longitude/Latitude (WGS 72) / 4322 Longitude/Latitude (WGS 84) / 4326 Longitude/Latitude (NAD 83) / 4269 Longitude/Latitude (OSGB 36) / 4277 British National Grid / 27700
CoordinateFormat	Specifies the format of latitude/longitude coordinates in the input.

optionName	Description/Valid Values
	Note: Use this option only if you specify a Latitude/Longitude coordinate system. If the coordinate system is not a Latitude/Longitude coordinate system, set the coordinate format to Decimal. One of the following: **Decimal** (90.000000, 180.000000) **DecimalAssumed** (90000000, 180000000). Default. **DegreesMinutesSeconds** (90 00 00N, 180 00 00W) **PreZero** (090000000N, 180000000W) **PreZeroDecimal** (090.000000N, 180.000000W)
DefaultTravelBoundaryCost	Number of cost units. The default is 10.
DefaultTravelBoundaryCostUnits	Type of metric you want to use to calculate the travel boundary. One of the following: • Feet • Hours • Hours • Kilometers (default) • Meters • Miles • Minutes • Seconds • Yards

Output Options

Table 185: GetTravelBoundary Output Options

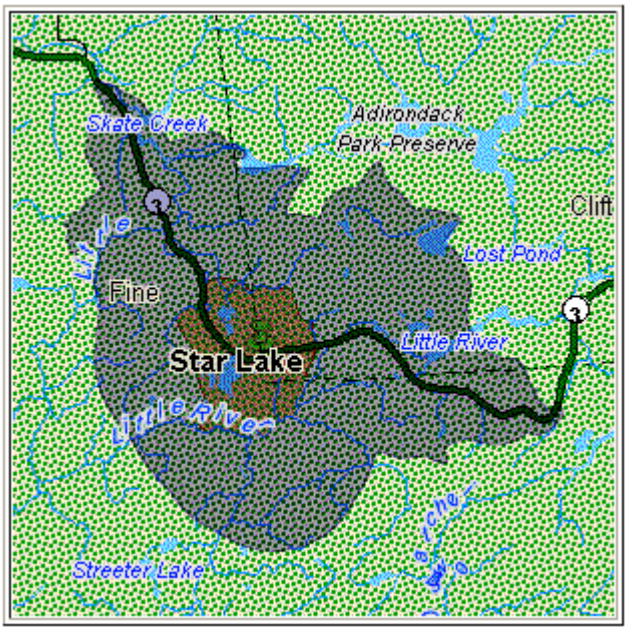
optionName	Description/Valid Values
ResultType	Specifies the type of result you want returned. One of the following: **AccessibleNodes** Returns the entire isoChrone. **Geometry** Returns all of the points along the road network that can be reached for the isoChrone calculation.

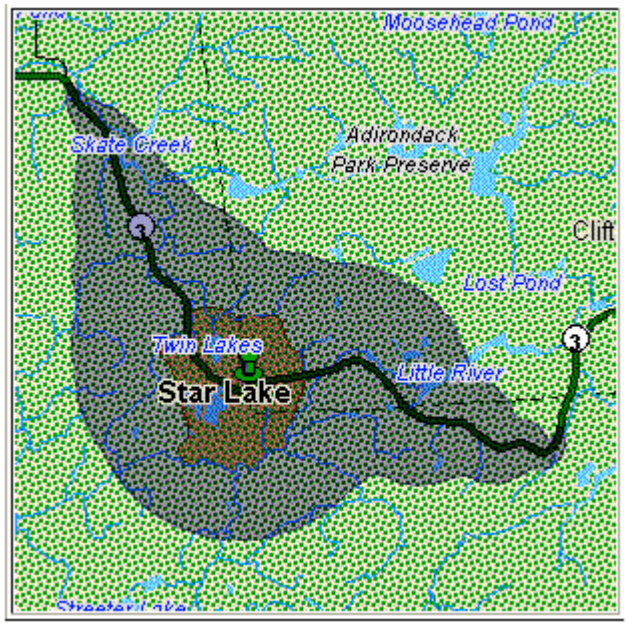
optionName	Description/Valid Values	
	StartNodes	Returns the location specified by the entered address.
SimplificationFactor	Specifies what percentage of the original points should be returned or upon which the resulting polygon should be based.	
BandingStyle	Specifies the style of banding to be used in the result. Banding styles are the types of multiple isoChrone or distance bands that can be displayed based on multiple costs.	
	Donut	Each boundary is determined by subtracting out the next smallest boundary.
	Encompassing	Each boundary is determined independent of all others.
ReturnHoles	Specifies whether you want to return holes, which are areas within the larger boundary that cannot be reached within the desired time or distance, based on the road network.	
	One of the following:	
	Y	Yes, return holes.
	N	Do not return holes. Default.
ReturnIslands	Specifies whether you want to return islands, which are small areas outside the main boundary that can be reached within the desired time or distance.	
	One of the following:	
	Y	Yes, return islands.
	N	Do not return islands. Default.

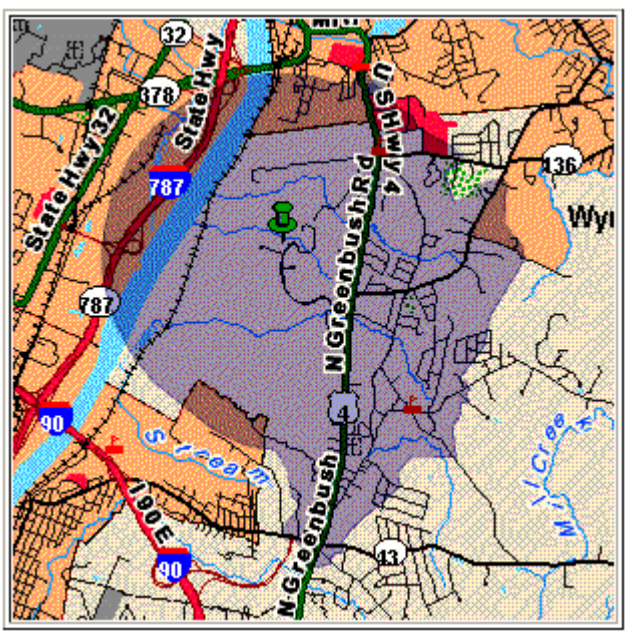
Travel Options

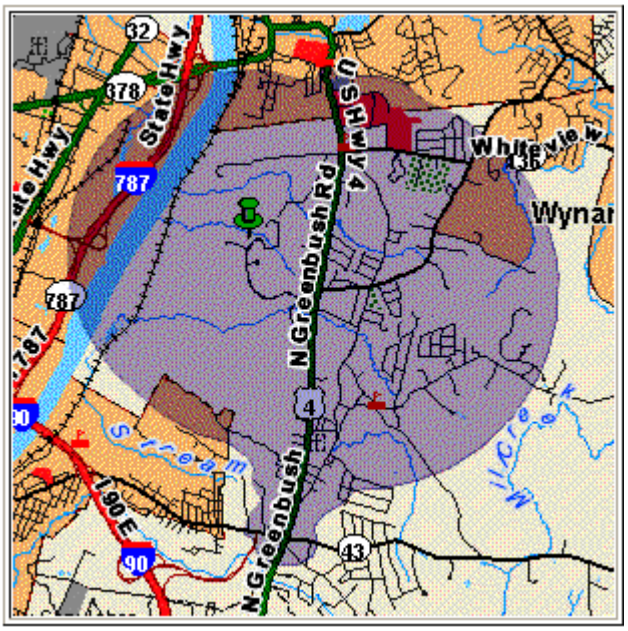
Travel options specify assumptions to make about travel speed off network roads and whether to use only major roads when calculating the travel boundary. Most travel options have to do with ambient speed.

Table 186: GetTravelBoundary Travel Options

optionName	Description/Valid Values
MaximumOffRoadDistance	Specifies the maximum distance to allow travel off the road network. Examples of off-network roads include driveways and access roads. For example, if you specify a maximum off road distance of 1 mile the travel boundary will extend no further than one mile from the network road. If you specify a value of 0 the travel boundary will not extend beyond the road itself. Use the ambient speed options to specify the speed of travel along non-network roads.
Units	The units of measure in which you want the data returned. One of the following: • Kilometer (default) • Meter • Mile
MajorRoads	Specifies whether to include all roads in the calculation or just major roads. If you choose to include only major roads, performance will improve but accuracy may decrease. This map represents a travel boundary with travel allowed on all roads: 

optionName	Description/Valid Values
	This map represents a travel boundary with travel restricted to major roads only:  One of the following: **Y** Include only major roads in the calculation. **N** Include all roads in the calculation. Default.
DefaultAmbientSpeed	Specifies the speed to travel when going off a network road to find the travel boundary. Examples of off-network travel include driveways and access roads. This option is available only when you specify a time value in the DefaultCostUnits option or the TravelBoundaryCostUnits input field. The default is 15. Specify the speed units in the AmbientSpeedUnit option. To control how off-network travel is used in the travel boundary calculation, you need to specify the speed of travel off the road network (the ambient speed). Ambient speed can affect the size and shape of the travel boundary polygon. In general, the faster the ambient speed, the larger the polygon. For example, if you were at a point with 5 minutes left, and if the ambient speed were 15 miles per hour, boundary points would be put at a distance of 1.25 miles. If the ambient speed were reduced to 10 miles per hour, boundary points would

optionName	Description/Valid Values
	be put at a distance of 0.83 miles. Note that you can limit the distance allowed off a network road by using the the MaximumOffRoadDistance option. **Note:** If you are calculating pedestrian travel boundaries we recommend that you change the default ambient speed to 3 MPH (5 KPH).
AmbientSpeedUnit	The unit of measure to use with the value specified in the DefaultAmbientSpeed option. One of the following: **KPH** Kilometers per hour. **MPH** MILES per hour. Default. **MTPS** Meters per second.
AmbientSpeed.RoadType.<Type>	Specifies the ambient speed to use for off-network travel based on the road type. If you do not specify an ambient speed for a road type, the default ambient speed will be used, as specified in the DefaultAmbientSpeed option. The following map shows an example of a travel boundary without ambient speed overrides:  For comparison, this map shows the same travel boundary with ambient speed overrides:

optionName	Description/Valid Values
	 <Type> can be one of the following: • AccessWay • Backroad • Connector • Ferry • Footpath • LimitedAccessDenseUrban • LimitedAccessRural • LimitedAccessSuburban • LimitedAccessUrban • LocalRoadDenseUrban • LocalRoadRural • LocalRoadSuburban • LocalRoadUrban • MajorLocalRoadDenseUrban • MajorLocalRoadRural • MajorLocalRoadSuburban • MajorLocalRoadUrban • MajorRoadDenseUrban • MajorRoadRural • MajorRoadSuburban • MajorRoadUrban

optionName	Description/Valid Values
	• MinorLocalRoadDenseUrban • MinorLocalRoadRural • MinorLocalRoadSuburban • MinorLocalRoadUrban • NormalRoadDenseUrban • NormalRoadRural • NormalRoadRural • NormalRoadUrban • PrimaryHighwayDenseUrban • PrimaryHighwayRural • PrimaryHighwaySuburban • PrimaryHighwayUrban • RampDenseUrban • RampLimitedAccess • RampMajorRoad • RampPrimaryHighway • RampRural • RampSecondaryHighway • RampUrban • RampSuburban • SecondaryHighwayDenseUrban • SecondaryHighwayRural • SecondaryHighwaySuburban • SecondaryHighwayUrban

Output

Get Travel Boundary returns the following fields:

Note: If you are using the API to interact with this service, the output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section of the *Spectrum™ Technology Platform API Guide*.

Table 187: GetTravelBoundary Outputs

optionName	Format	Description / Valid Values
Status	String	Reports the success or failure of the match attempt. **null** Success

optionName	Format	Description / Valid Values
		F Failure
Status.Code	String	Reason for failure, if there is one. One of the following: • InsufficientInputData (missing lat/lon) • MalformedInputData (wrong input format) • InputOutOfRange (input is out of range) • EngineError (engine-generated error)
Status.Description	String	Description of failure indicated in Status.Code.

GetTravelCostMatrix

GetTravelCostMatrix calculates the travel time and distances between an array of start and end locations. You can use GetTravelCostMatrix to find the shortest or fastest paths between a number of start points and a number of end points, and determine the total time and distance of the individual routes (the route costs). For example if you input four start points (S1 through S4) and four end points (E1 through E4), a total of 16 routes will be returned as illustrated in the following diagram:

Matrix routing is often used to determine service response time and coverages for specific services such as a fire house or police station. You may require these calculations to ensure they can provide adequate coverage for service level agreements such as having one or more people who can respond to an incident within 20 minutes of first notification. The calculation can be used an analysis tool to determine the risk of an insured property or person based on the probability that ambulance, public safety, or fire personnel can reach the property/person in a reasonable amount of time.

Note: If you intend to interact with Get Travel Cost Matrix as a service, note that it is only available as a web service. Get Travel Cost Matrix is not available through the Java, C++, C, .NET, or COM APIs.

Input

The input to Get Travel Cost Matrix is a list of start points and end points expressed as latitude/longitude coordinates. You can also include user-defined fields.

The order of start and end points in the input determines the order of the sequences in the response. For example, if you specify two start points and two end points in the request, the response will contain the following order of sequences (where S is start and E is end): S1 to E1, S1 to E2, S2 to E1, S2 to E2.

Note: If you intend to interact with the Get Travel Cost Matrix service, note that it is only available as a web service. The Get Travel Cost Matrix is not available through the Java, C++, C, .NET, or COM APIs.

Get Travel Cost Matrix input is formatted according to the structure defined in the Get Travel Cost Matrix WSDL. You can access the WSDL on your Spectrum™ Technology Platform server at:

http://<servername>:<port>/services/GetTravelCostMatrix?wsdl

The input start and end points are structured as follows.

```
<ns:rows>
    <ns:RouteMatrixRequest>
        <ns:StartPoints>
            <ns:StartPoint>
                <ns:Latitude><varname>value</varname></get:Latitude>
                <ns:Longitude><varname>value</varname></get:Longitude>
            </ns:StartPoint>
        </ns:StartPoints>
        <ns:EndPoints>
            <ns:EndPoint>
                <ns:Latitude><varname>value</varname></get:Latitude>
                <ns:Longitude><varname>value</varname></get:Longitude>
            </ns:EndPoint>
        </ns:EndPoints>
        <ns:user_fields>
            <ns:user_field>
                <ns:name><varname>value</varname></get:name>
                <ns:value><varname>value</varname></get:value>
            </ns:user_field>
        </ns:user_fields>
    </ns:RouteMatrixRequest>
</ns:rows>
```

When using Get Travel Cost Matrix in a job you must convert flat data to a list. This is because input data for a job is flat data read from a file or database but Get Travel Cost Matrix requires input to be in list format. To convert flat data to a list, use the Transformer stage to create a custom transform.

Table 188: Get Travel Cost Matrix Input

Element Name	Description / Valid Values
Latitude	The latitude for a start or end point. Specify the latitude using the format selected in the CoordinateFormat option.
Longitude	The longitude for a start or end point. Specify the longitude using the format selected in the CoordinateFormat option.

Element Name	Description / Valid Values
ID	An identification you assign to the point. Specify an ID comprised of alpha and/or numeric characters to represent a point. This ID corresponds to the StartPointID or EndPointID field in output.

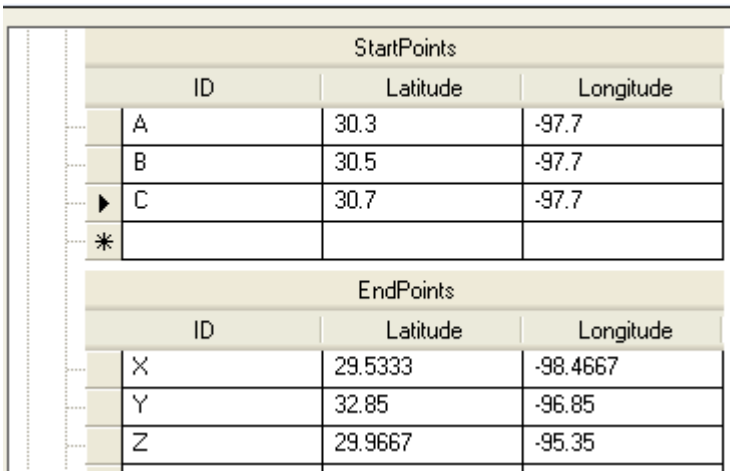
Sample Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:get="http://www.g1.com/services/GetTravelCostMatrix">
    <soapenv:Header/>
    <soapenv:Body>
        <get:GetTravelCostMatrixRequest>
            <get:context>
                <get:account.id>guest</get:account.id>
                <get:account.password></get:account.password>
            </get:context>
            <get:rows>
                <get:RouteMatrixRequest>
                    <get:StartPoints>
                        <get:StartPoint>
                            <get:Latitude>41.857333</get:Latitude>
                            <get:Longitude>-88.325183</get:Longitude>
                        </get:StartPoint>
                        <get:StartPoint>
                            <get:Latitude>41.881833</get:Latitude>
                            <get:Longitude>-87.785587</get:Longitude>
                        </get:StartPoint>
                    </get:StartPoints>
                    <get:EndPoints>
                        <get:EndPoint>
                            <get:Latitude>42.009305</get:Latitude>
                            <get:Longitude>-87.695284</get:Longitude>
                        </get:EndPoint>
                        <get:EndPoint>
                            <get:Latitude>42.779007</get:Latitude>
                            <get:Longitude>-86.139898</get:Longitude>
                        </get:EndPoint>
                    </get:EndPoints>
                    <get:user_fields>
                        <get:user_field>
                            <get:name>TravelDate</get:name>
                            <get:value>12-28-2009</get:value>
                        </get:user_field>
                    </get:user_fields>
                </get:RouteMatrixRequest>
            </get:rows>
        </get:GetTravelCostMatrixRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Sample Input

The sample input below shows three start points and three end points. The start points have been given IDs of A, B, and C, and the end points have been given IDs of X, Y, and Z. Click **here** to see how these IDs correlate to the output.

Preview Input



StartPoints

ID	Latitude	Longitude
A	30.3	-97.7
B	30.5	-97.7
C	30.7	-97.7

EndPoints

ID	Latitude	Longitude
X	29.5333	-98.4667
Y	32.85	-96.85
Z	29.9667	-95.35

Options

Routing Options

Table 189: Routing Options

optionName	Description
DataSetResourceName	The name of the database that contains the data to use in the search process. Use the database name specified in the Enterprise Routing Module Routing Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.
OptimizeBy	Specifies whether to find the shortest distance or the shortest time. One of the following: **Time** Optimize by fastest time traveled. Default. **Distance** Optimize by shortest distance traveled.
ReturnOptimalRoutesOnly	Specifies whether to return only the optimized route for each start point/end point combination. The optimized route is either the fastest route or the shortest distance, depending on the selection you make for the OptimizeBy option. For example, if you have one start point (S1) and two end points (E1 and E2) and it is faster to go from S1 to E2 than from S1 to E1,

<table>
<tr><th>optionName</th><th>Description</th></tr>
<tr><td></td><td>and you choose to return only the optimal route, then only the route cost information for S1 to E2 will be returned.

One of the following:

Y Yes, return only optimal routes. Default.

N No, return all routes not just the optimal routes.</td></tr>
<tr><td>RouteCostMatrixFormat</td><td>Specifies the data format of the route cost matrix. One of the following:

Hierarchy The route cost matrix is returned as list that you can view in Management Console, Enterprise Designer, or Interactive Driver. Choose this option if you want to view the data in these tools or if you want to work with the data in tabular format. Default.

Object The route cost matrix is returned as a data object that you cannot interact with directly. Only select this option if the output is going directly to another stage in a dataflow or a web service call, or if it will be used by another application to visualize the routes, such as input to mapping software.</td></tr>
<tr><td>CoordinateSystem</td><td>The coordinate system of the input coordinates. The format must be the European Petroleum Survey Group (EPSG) code, the SRID code, or the MAPINFO code. Specify the coordinate reference system in the format <code>codespace:code</code>. For example, a geometry in Latitude/Longitude (WGS 84) would be <code>EPSG:4326</code>. The default value is EPSG:4326, which is also known as Latitude/Longitude (WGS 84). This is a worldwide system and is the terrestrial reference frame associated with the NAVSTAR Global Positioning System (GPS) used extensively for navigation and surveying. This datum is earth-centered ("geocentric") and defined with satellite and terrestrial data.

The following table lists some of the more widely used EPSG codes:

<table>
<tr><th>Coordinate System</th><th>EPSG Code</th></tr>
<tr><td>Longitude/Latitude (WGS 72)</td><td>4322</td></tr>
<tr><td>Longitude/Latitude (WGS 84)</td><td>4326</td></tr>
<tr><td>Longitude/Latitude (NAD 83)</td><td>4269</td></tr>
<tr><td>Longitude/Latitude (OSGB 36)</td><td>4277</td></tr>
<tr><td>British National Grid</td><td>27700</td></tr>
</table>
</td></tr>
<tr><td>CoordinateFormat</td><td>Specifies the format of latitude/longitude coordinates in the input.</td></tr>
</table>

optionName	Description
	Note: Use this option only if you specify a Latitude/Longitude coordinate system. If the coordinate system is not a Latitude/Longitude coordinate system, set the coordinate format to Decimal. One of the following: **Decimal** (90.000000, 180.000000). Default. **DecimalAssumed** (90000000, 180000000). **DegMinSec** (90 00 00N, 180 00 00W) **PreZero** (090000000N, 180000000W) **PreZeroDecimal** (090.000000N, 180.000000W)
DistanceUnits	Specifies how GetTravelCostMatrix should return distance values. One of the following: **Feet** Return distance in feet. **Kilometers** Return distance in kilometers. **Meters** Return distance in meters. **Miles** Return distance in miles. Default. **Yards** Return distance in yards.
TimeUnits	Specifies how to return time. One of the following: **Hours** Return time in hours. **Minutes** Return time in minutes. Default. **Seconds** Return time in seconds.

Travel Options

This set of preferences allows you to set the desirability for each road type. For instance, you can request that the server attempt to avoid all of the major road types.

Table 190: Travel Preferences Options

optionName	Description/Valid Values
RoadType_<type>	Specifies the priority to give to different types of roads when determining the route.

optionName	Description/Valid Values
	Each road type has its own option, named `RoadType_<type>`, where <type> is one of the following: • AccessWay • Backroad • Connector • Ferry • Footpath • LimitedAccessDenseUrban • LimitedAccessRural • LimitedAccessSuburban • LimitedAccessUrban • LocalRoadDenseUrban • LocalRoadRural • LocalRoadSuburban • LocalRoadUrban • MajorLocalRoadDenseUrban • MajorLocalRoadRural • MajorLocalRoadSuburban • MajorLocalRoadUrban • MajorRoadDenseUrban • MajorRoadRural • MajorRoadSuburban • MajorRoadUrban • MinorLocalRoadDenseUrban • MinorLocalRoadRural • MinorLocalRoadSuburban • MinorLocalRoadUrban • NormalRoadDenseUrban • NormalRoadRural • NormalRoadSuburban • NormalRoadUrban • PrimaryHighwayDenseUrban • PrimaryHighwayRural • PrimaryHighwaySuburban • Primary Highway Urban • PrimaryHighwayUrban • RampDenseUrban • RampLimitedAccess • RampMajorRoad • RampPrimaryHighway

optionName	Description/Valid Values
	• RampRural • RampSecondaryHighway • RampSuburban • RampUrban • SecondaryHighwayDenseUrban • SecondaryHighwayRural • SecondaryHighwaySuburban • SecondaryHighwayUrban For each road type you can specify one of the following: **Avoid** Exclude the road type from routes if possible. **Note:** It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment. **High** Prefer the road type over other road types. **Low** Prefer other road types over this road type. **Medium** Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium.

Output

The output from GetTravelCostMatrix contains a number of route sequences, each containing the start and end points for the candidates used in the matrix route as well as the time and distance for each route that has been calculated. The output may contain all routes in the matrix, or only the optimal route between each start/end point, depending on how you configure the OptimizeBy option.

If GetTravelCostMatrix cannot find a route between a start and end point in the matrix, an error will be logged in the server log but the routes that could be determined will be returned. For example, if you have start point A and end points 1 and 2, and GetTravelCostMatrix can find a route from A to 1 but not from A to 2, GetTravelCostMatrix will return the route from A to 1 and log an error that the route from A to 2 could not be determined.

GetTravelCostMatrix output is either in list format or object format depending on how you configure the ReturnRouteCostMatrix option. You cannot interact directly with object output.

List output is structured as follows:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <ns2:GetTravelCostMatrixResponse
xmlns:ns2="http://www.g1.com/services/GetTravelCostMatrix">
            <ns2:rows>
                <ns2:RouteMatrixResponse>
                    <ns2:TimeUnits><varname>value</varname></ns2:TimeUnits>

<ns2:DistanceUnits><varname>value</varname></ns2:DistanceUnits>
                    <ns2:RouteCosts>
                        <ns2:RouteCost>

<ns2:StartPointRef><varname>value</varname></ns2:StartPointRef>

<ns2:EndPointRef><varname>value</varname></ns2:EndPointRef>
                            <ns2:Time><varname>value</varname></ns2:Time>

<ns2:Distance><varname>value</varname></ns2:Distance>
                        </ns2:RouteCost>
                        <ns2:RouteCost>

<ns2:StartPointRef><varname>value</varname></ns2:StartPointRef>

<ns2:EndPointRef><varname>value</varname></ns2:EndPointRef>
                            <ns2:Time><varname>value</varname></ns2:Time>

<ns2:Distance><varname>value</varname></ns2:Distance>
                        </ns2:RouteCost>
                    </ns2:RouteCosts>
                </ns2:RouteMatrixResponse>
            </ns2:rows>
        </ns2:GetTravelCostMatrixResponse>
    </soap:Body>
</soap:Envelope>
```

Table 191: Get Travel Cost Matrix Output

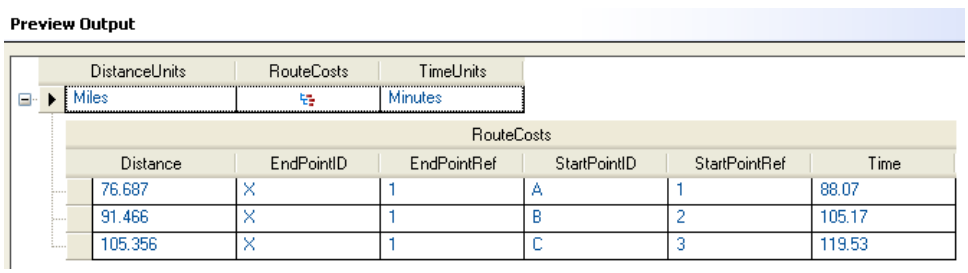
Element Name	Format	Description / Valid Values
Distance	String	The distance from the start point to the end point. The value is in the units indicated in the DistanceUnits element.
DistanceUnits	String	The unit of measurement used for distance. One of the following: **Feet** The distance is in feet. **Kilometers** The distance is in kilometers. **Meters** The distance is in meters. **Miles** The distance is in miles. Default.

Element Name	Format	Description / Valid Values
		Yards The distance is in yards.
EndPointRef	String	A reference ID that corresponds to the order in which the end points were specified in the input. The first end point specified has a reference ID of 1, the second has an ID of 2, and so on. You must develop your own process for associating the latitude/longitude coordinates in the input with the reference ID returned by Get Travel Cost Matrix.
EndPointID	String	An identification you assigned to the corresponding end point in the ID field of the Input stage. For example, the first end point could have an EndPointID of N, the second end point could have an EndPointID of O, and so on. **Note:** This field is active only when the **Route Cost Matrix Format** field is set to "Hierarchy."
StartPointRef	String	A reference ID that corresponds to the order in which the start points were specified in the input. The first start point specified has a reference ID of 1, the second has an ID of 2, and so on. You must develop your own process for associating the latitude/longitude coordinates in the input with the reference ID returned by Get Travel Cost Matrix.
StartPointID	String	An identification you assigned to the corresponding start point in the ID field of the Input stage. For example, the first start point could have a StartPointID of A, the second start point could have a StartPointID of B, and so on. **Note:** This field is active only when the **Route Cost Matrix Format** field is set to "Hierarchy."
Time	String	The total time from the start point to the end point. The value is in the units indicated in the TimeUnits element.
TimeUnits	String	The unit of measurement used for time. One of the following: **Hours** The time is in hours. **Minutes** The time is in minutes. Default. **Seconds** The time is in seconds.

Element Name	Format	Description / Valid Values
Status	String [1]	Reports the success or failure of the match. **null** Success **F** Failure
Status.Code	String [100]	Reason for failure, if there is one.
Status.Description	String	A description of failure indicated in Status.Code.

Sample Output

The sample output below shows that the StartPointIDs assigned in the sample **input** (named A, B, and C upon input) correspond to the first three entries in the list of points (numbered 1, 2, and 3 respectively by Spectrum™ Technology Platform). It also shows that the first EndPointID (named X upon input) is the most efficient route from all three StartPointIDs, as compared to the other two EndPointIDs (named Y and Z upon input, not shown below because EndPointID X is the more efficient route).



GetTravelDirections

GetTravelDirections is a tool that allows you to return routing information for a set of two distinct points or for multiple points. It takes a starting latitude and longitude point and an ending latitude and longitude point as input and returns the route that is either fastest or shortest, depending on how you configure the stage.

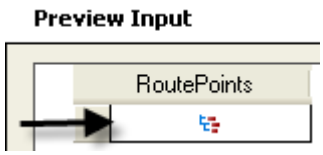
Each country has its own database, named in this format: Enterprise Routing Module - <Country>. Each database also has its own country code. For example, the name of the Austrian database is "Enterprise Routing Module - Austria," and the Austrian batch country code is "A1T." Each database requires a separate license.

Note: If you use the API to interact with this service, note that this service is available only via web services, not through any other API language.

GetTravelDirections is part of the Enterprise Routing Module. For more information on this module, including a listing of other components included with it, see **Enterprise Routing Components** on page 614.

To familiarize yourself with how GetTravelDirections works, use the Management Console's Preview tab to step through the following procedure.

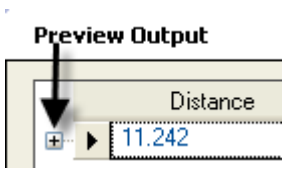
1. Open the Management Console.
2. Under the Services node, select `Get Travel Directions`.
3. On the **Options** tab, specify the options you want. For information on the options, see **Options** on page 636.
4. Click the **Preview** tab.
5. Click inside the RoutePoints cell:



6. Enter latitude information into the Latitude cell and longitude information into the Longitude cell. Alternatively, you can paste this information into the cell, even in multiples. If you have a two-column set of data separated by tabs, you can highlight the data and paste it into the first Latitude cell. It will populate as many rows (of both latitude and longitude) as your data requires.

   Note: The Import Data and View buttons are grayed out because you cannot import data or change the view from horizontal to vertical in a hierarchical structure.

7. Click **Run Preview**.
8. Under Preview Output, expand the data by clicking the plus sign.



The Route Directions appear.

Input

GetTravelDirections takes route points, which include starting and ending latitudes and longitudes as input. The following table provides information on the format and layout of the input.

Note: If you intend to interact with the GetTravelDirections service, note that it is only available as a web service. The Get Travel Directions service is not available through the Java, C++, C, .NET, or COM APIs.

GetTravelDirections input is formatted according to the structure defined in the Get Travel Directions WSDL. You can access the WSDL on your Spectrum™ Technology Platform server at:

http://<servername>:<port>/services/GetTravelDirections?wsdl

Table 192: GetTravel Directions Input Data

columnName	Format	Description / Valid Values
RoutePoints	List	A schema containing the latitude string and the longitude string. For information on creating a process list, see the *Spectrum™ Technology Platform User's Guide*.
Language	String	Specifies the language in which GetTravelDirections should return directions. One of the following: **da** Return directions in Danish. **nl** Return directions in Dutch. **en** Return directions in English. Default **en_US** Return directions in American English. **fi** Return directions in Finnish. **fr** Return directions in French. **de** Return directions in German. **it** Return directions in Italian. **jp** Return directions in Japanese. **no** Return directions in Norwegian. **pt** Return directions in Portuguese. **es** Return directions in Spanish. **sv** Return directions in Swedish. **Note:** An entry in this input field will override an entry in the Default Language option.

Options

GetTravelDirections allows you to set default processing options through the Management Console graphical user interface. You can override certain settings for individual calls using the Client API or Spectrum™ Technology Platform client tools, such as Enterprise Designer.

If you do not specify system-level defaults through the Management Console or a custom interface, and GetTravelDirections does not pass any processing options at the time of the call, Spectrum™ Technology Platform uses the default options listed in the following sections.

Routing

The following table lists the configuration options for GetTravelDirections.

Table 193: Get Travel Directions Configuration Options

<table>
<tr><th>optionName</th><th>Description/Valid Values</th></tr>
<tr><td>DataSetResourceName</td><td>The name of the database that contains the data to use in the search process. Use the database name specified in the Management Console Database Resource tool. For more information, see the Spectrum™ Technology Platform User's Guide.</td></tr>
<tr><td>OptimizeBy</td><td>Specifies whether GetTravelDirections should find the shortest distance or the shortest time.
One of the following:
Time Optimize by shortest time traveled. Default.
Distance Optimize by shortest distance traveled.</td></tr>
<tr><td>CoordinateSystem</td><td>The coordinate system of the input coordinates. The format must be the European Petroleum Survey Group (EPSG) code, the SRID code, or the MAPINFO code. Specify the coordinate reference system in the format <code>codespace:code</code>. For example, a geometry in Latitude/Longitude (WGS 84) would be <code>EPSG:4326</code>. The default value is EPSG:4326, which is also known as Latitude/Longitude (WGS 84). This is a worldwide system and is the terrestrial reference frame associated with the NAVSTAR Global Positioning System (GPS) used extensively for navigation and surveying. This datum is earth-centered ("geocentric") and defined with satellite and terrestrial data.
The following table lists some of the more widely used EPSG codes:
<table>
<tr><th>Coordinate System</th><th>EPSG Code</th></tr>
<tr><td>Longitude/Latitude (WGS 72)</td><td>4322</td></tr>
<tr><td>Longitude/Latitude (WGS 84)</td><td>4326</td></tr>
<tr><td>Longitude/Latitude (NAD 83)</td><td>4269</td></tr>
<tr><td>Longitude/Latitude (OSGB 36)</td><td>4277</td></tr>
<tr><td>British National Grid</td><td>27700</td></tr>
</table>
</td></tr>
</table>

optionName	Description/Valid Values
CoordinateFormat	Specifies the format of latitude/longitude coordinates in the input. **Note:** Use this option only if you specify a Latitude/Longitude coordinate system. If the coordinate system is not a Latitude/Longitude coordinate system, set the coordinate format to Decimal. One of the following: **Decimal** (90.000000, 180.000000). Default. **DecimalAssumed** (90000000, 180000000). **DegreesMinutesSeconds** (90 00 00N, 180 00 00W) **PreZero** (090000000N, 180000000W) **PreZeroDecimal** (090.000000N, 180.000000W)
DirectionsStyle	Specifies how directions should be returned. One of the following: **Normal** Return directions in a normal format. Default. **Terse** Return directions in a terse format. Terse directions are more suitable for wireless devices.
DistanceUnits	Specifies how GetTravelDirections should return distance values. One of the following: **Feet** Return distance in feet. **Kilometers** Return distance in kilometers. **Meters** Return distance in meters. **Miles** Return distance in miles. Default. **Yards** Return distance in yards.
TimeUnits	Specifies how GetTravelDirections should return time. One of the following: **Hours** Return time in hours. **Minutes** Return time in minutes. Default. **Seconds** Return time in seconds.

optionName	Description/Valid Values
FocusOfRoute	Specifies the focus of the route. A focused route is a subset of the whole route that concentrates on either the beginning or end of the route. A route focused at the start will route the user from their origin to (and including) the first major highway. A route focused at the end will route the user from the last major highway in the route (including that highway) to the destination. If there are no major highways in the route, the focused route will be the same as an unfocused route. One of the following: **Start** Return just the start of the route. **End** Return just the end of the route. **None** Return the whole route. Default.
OptimizeIntermediatePoints	Specifies whether GetTravelDirections re-orders the intermediate points in an optimal manner during route computation. One of the following: **Y** Yes, optimize intermediate points. Directions optimally orders the intermediate points. Default. **N** No, do not optimize intermediate points. Directions preserves the specified order of the points.

Directions

Table 194: Get Travel Directions Direction Options

optionName	Description/Valid Values
ShowDistance	Specifies whether to return the distance for the route. One of the following: **Y** Yes, return the distance for the route. Default. **N** No, do not return the distance for the route.
ReturnRouteDirections	Specifies whether to return turn-by-turn directions for the route. This option is enabled by default. One of the following: **Y** Yes, return textual turn-by-turn directions. Default.

optionName	Description/Valid Values
	N No, do not return textual turn-by-turn directions.
ReturnSegmentGeometry	Specifies whether to return a set of latitude/longitude points that represent a geometry for a segment within a route. The segment geometry is used to create the route geometry. The output of this field is shown within the RouteDirections output field. For more information about route geometry, see **What is Route Geometry?** on page 641.
ReturnRouteGeometry	Specifies whether to return a set of latitude/longitude points that represent a route geometry. The route geometry can be used to create a route map and to perform analysis on the route. One of the following: **All** Return all points in the route geometry. **End** Return just the end points of each route segment geometry. **None** Do not return route geometry. Default.
DefaultLanguage	Specifies the language in which GetTravelDirections should return directions. One of the following: **da** Return directions in Danish. **nl** Return directions in Dutch. **en** Return directions in English. Default **en_US** Return directions in American English. **fi** Return directions in Finnish. **fr** Return directions in French. **de** Return directions in German. **it** Return directions in Italian. **jp** Return directions in Japanese. **no** Return directions in Norwegian. **pt** Return directions in Portuguese. **es** Return directions in Spanish. **sv** Return directions in Swedish.
ShowTime	Specifies whether GetTravelDirections should return the time it takes to follow a direction within the route. One of the following:

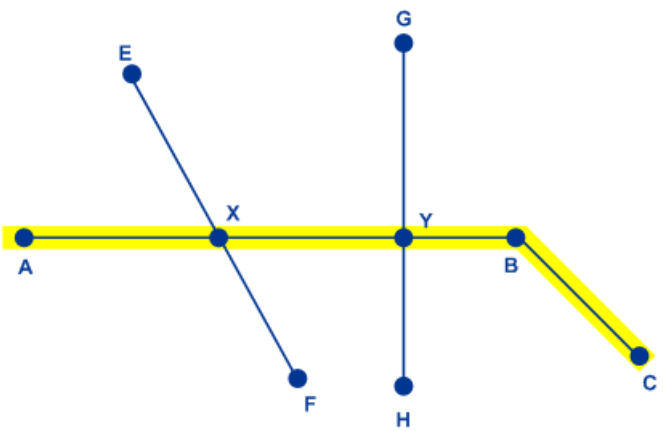
optionName	Description/Valid Values
	Y Yes, return the time for the route. Default. **N** No, do not return the time for the route.
ShowPrimaryNameOnly	Specifies whether GetTravelDirections should return all names for a given street in the directions or if it should return just the primary name for a street. One of the following: **Y** Yes, return just the primary name. **N** Return all names for a street. Default.

What is Route Geometry?

A route geometry is a series of latitude/longitude points that form a route. A route geometry can be as simple as a single point, such as a route that starts and ends on the same street segment:

Since the starting point is always known the route geometry in this very simple example could be just the end point. So if A is the starting point, the route geometry could be the latitude/longitude of point B.

In a more complex route geometry involving multiple route segments there can be several points in the route geometry. Consider the route highlighted below which starts at point A and ends at point C, traveling through intersections X, Y, and B:



In this route, the full route geometry would consist of the latitude/longitude of points A, X, Y, B, and C. Note that you can control which points to actually return and may choose to include all points in the route geometry or just the end points of each route segment. In the above example, the end points are B and C, since A to B is one route segment and B to C is another.

Travel Options

This set of preferences allows you to set the desirability for each road type. For instance, you can request that the server attempt to avoid all of the major road types.

Table 195: Travel Preferences Options

optionName	Description/Valid Values
RoadType_<type>	Specifies the priority to give to different types of roads when determining the route. Each road type has its own option, named `RoadType_<type>`, where <type> is one of the following: • AccessWay • Backroad • Connector • Ferry • Footpath • LimitedAccessDenseUrban • LimitedAccessRural • LimitedAccessSuburban • LimitedAccessUrban • LocalRoadDenseUrban • LocalRoadRural • LocalRoadSuburban • LocalRoadUrban • MajorLocalRoadDenseUrban • MajorLocalRoadRural • MajorLocalRoadSuburban • MajorLocalRoadUrban • MajorRoadDenseUrban • MajorRoadRural • MajorRoadSuburban • MajorRoadUrban • MinorLocalRoadDenseUrban • MinorLocalRoadRural • MinorLocalRoadSuburban • MinorLocalRoadUrban • NormalRoadDenseUrban • NormalRoadRural • NormalRoadSuburban • NormalRoadUrban

optionName	Description/Valid Values
	• PrimaryHighwayDenseUrban • PrimaryHighwayRural • PrimaryHighwaySuburban • Primary Highway Urban • PrimaryHighwayUrban • RampDenseUrban • RampLimitedAccess • RampMajorRoad • RampPrimaryHighway • RampRural • RampSecondaryHighway • RampSuburban • RampUrban • SecondaryHighwayDenseUrban • SecondaryHighwayRural • SecondaryHighwaySuburban • SecondaryHighwayUrban For each road type you can specify one of the following: **Avoid** Exclude the road type from routes if possible. **Note:** It is not always possible to exclude a road type from the travel directions. Depending on the situation, the alternative to an avoided road type may be so poor that the software will choose a route that uses an avoided road type. Also, if the starting or ending point lies along a segment whose road type has been avoided, the software will still use that segment. **High** Prefer the road type over other road types. **Low** Prefer other road types over this road type. **Medium** Give this road type equal preference with other road types. If no preference is specified for a road type, the default is Medium.

Output

GetTravelDirections returns the following fields:

Note: If you are using the API to interact with this service, The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section of the *Spectrum™ Technology Platform API Guide*.

Table 196: GetTravelDirections Outputs

Field Name	Format	Description / Valid Values
Distance	String	The distance of each segment in the route.
DistanceUnits	String	The unit of measurement for distance.
Format	String	Value of the format used to generate directions.
Language	String	Language of the directions.
RouteDirections	List	Turn-by-turn directions for the route.
RouteGeometry	Geometry object	A geometry object containing the coordinates of each point in the route. For more information, see **What is Route Geometry?** on page 641 .
Status	String	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	String	Reason for failure, if there is one. One of the following: • InsufficientInputData(missing lat/lon) • MalformedInputData(wrong input format) • InputOutOfRange (input is out of range) • EngineError (engine generated error)
Status.Description	String	Description of failure indicated in Status.Code.
Time	String	Total time it takes to follow the route in minutes.
TimeUnits	String	The unit of measurement for time.

Routing Database Resources

Enterprise Routing Module database resources are used by Find Nearest, Get Travel Boundary, Get Travel Cost Matrix, and Get Travel Directions.

Adding a Routing Database Resource

1. Open the Management Console.
2. In Management Console, expand **Modules** > **Enterprise Routing** > **Tools** > **Routing Database Resources**.
3. Click **Add** to install a new database or click **Modify** to change an existing database resource.
4. If you are adding a new database, enter a name for the database resource in the **Name** field. The name can be anything you choose. You cannot modify the name of an existing database resource. Changing the name of an existing database resource would cause any services or jobs that reference the database resource to fail.
5. In the **Path elements** field, specify the folder that contains the database files. The database files can be found in: `<InstallLocation>\ERM-<CountryCode>\<Vintage>\<Type>\<Country>` where <InstallLocation> is the directory you specified when you installed the database files, <CountryCode> is the two-letter country code, <Vintage> is the date of the data (for example, 2009.1), <Type> is driving or pedestrian, and <Country> is the name of the country.
6. Click **OK**.

The database resource now appears in the list of available database resources for Find Nearest, Get Travel Boundary, Get Travel Cost Matrix, and Get Travel Directions.

Deleting a Routing Database

1. Open the Management Console.
2. Expand **Modules** > **Enterprise Routing** > **Tools** then click **Routing Database Resources**.
3. Highlight the routing database you want to remove and click **Delete**. You will receive a confirmation message asking if you wish to delete the database.
4. Click **OK** to confirm.

14

Enterprise Tax Module

In this section:

- What Is the Enterprise Tax Module? 648
- AssignGeoTAXInfo 651
- CalculateDistance 694
- Creating a User-Defined Auxiliary File 696

What Is the Enterprise Tax Module?

The Enterprise Tax Module determines the tax jurisdiction for an address. The Enterprise Tax Module takes a standardized address and matches it to an exact physical location, returning latitude/longitude coordinates with the correct place code for the address. This solution greatly reduces the inaccuracies associated with 9-digit and 5-digit ZIP Code-based matching.

The Enterprise Tax Module uses a database of tax jurisdictions provided by Tele Atlas North America. This data, which is collected through an ongoing research program and updated regularly, provides current jurisdictional boundary information down to the municipal and special tax district levels.

The Enterprise Tax Module can also calculate latitude/longitude coordinates for individual address locations, including the use of interpolation and offset.

Note: The Enterprise Tax Module processes only U.S. addresses. Before using the Enterprise Tax Module, you need to standardize your address data using a product such as the Universal Addressing Module.

Enterprise Tax Components

The Enterprise Tax Module consists of the following components.

- **AssignGeoTAXInfo** - Takes an input address and returns census, latitude/longitude, and tax information about the address. AssignGeoTAXInfo utilizes Pitney Bowes Software's GeoTAX technology.
- **CalculateDistance** - Takes two latitude/longitude coordinates as input and computes and returns the distance between the coordinates.

Enterprise Tax Databases

The Enterprise Tax Module provides you with several different databases, along with the ability to include additional databases to match against your input addresses.

Table 197: Enterprise Tax Module Databases

Database Name & Description	Required or Optional	Supplier
GeoTAX Master Files The master files are the main data files used by the Enterprise Tax module. They identify all geographic components associated with a street address, such as the latitude/longitude, census tract, and block group. This file, at over two gigabytes of data, is significantly larger than the postal file, but provides the greatest coding accuracy. The master files are on the disc labeled GeoTAX Subscription.	Required	Pitney Bowes Software monthly subscription

Database Name & Description	Required or Optional	Supplier
State-Supplied Files State-supplied files are provided by individual state governments that the Enterprise Tax Module uses to override results from the master files. The Enterprise Tax Module provides you with the ability to override, at the state level, match results based upon information supplied by the states. By matching to the state-supplied files, you can remain compliant with tax jurisdiction assignment requirements mandated by new federal and state laws, such as the Mobile Telecommunications Sourcing Act and the Florida state Communications Services Tax Simplification Law. Currently, there are two file formats supported in the Enterprise Tax Module: the Florida-native format, and the national TS-158 format (ANSI Transaction Set No. 158). The state of Florida provides address files in both the TS-158 and its own native format. The state of Washington provides address data in the TS-158 format. **Note:** This database option may not be available to all Enterprise Tax users. Individual states may restrict the use of state-supplied address files to licensed communications carriers or other business entities registered with the individual state. The Enterprise Tax Module first attempts to match to the state database. If the Enterprise Tax Module cannot find a state match, it attempts a match to the master files.	Optional	State Governments
GeoTAX Auxiliary File The GeoTAX Auxiliary file contains new addresses that have not yet been added to the Master File. It provides the most up-to-date address data possible.	Optional	Pitney Bowes Software monthly subscription
User Auxiliary File User Auxiliary files are user-defined files that the Enterprise Tax Module uses to override results from the master files in street-level matching. If you have data that is more current than that in the master files, you can enter the new data into the auxiliary file and use it for your address matching. The Enterprise Tax Module returns matches made with a code that signifies the answer came from the auxiliary file. You can also return user-defined data from the auxiliary file with the match.	Optional	User-defined

Database Name & Description	Required or Optional	Supplier
Boundary Files Boundary files provide additional data about locations of special tax districts: Special Purpose Tax Districts (SPDs), Insurance Premium Tax Districts (IPDs), Payroll Tax Districts (PAYs), and Personal Property Tax Districts (PTDs). • The **Special Purpose District (SPD) file** provides you with return data on special purpose tax districts. Special purpose tax districts include such districts as Regional Transit Areas and Metropolitan Football districts. • The **Insurance Premium District (IPD) file** is used by the insurance industry to determine sales tax on insurance premiums written in some states. This file allows insurers to correctly determine the rate due on each insurance policy. Boundaries vary by state and are based on fire and police district and municipal boundaries. • The **Payroll Tax District (PAY) file** can help your company comply with state legislation that requires employers to deduct taxes from employee paychecks for special districts, such as taxes for emergency municipal services districts. • The **Personal Property Tax District (PTD)** file provides your company with a complete solution for accurate, automated asset collection and jurisdiction assignment. Pitney Bowes Software provides you with the appropriate boundary file on separate media if you license any of the optional files. **Note:** The Enterprise Tax Module only loads one boundary file at a time. For more information, see the configuration options in **AssignGeoTAXInfo** on page 651.	Optional	Pitney Bowes Software
User-Defined Boundary Files A user-defined boundary file is a file that you create to represent polygons that you want to match against, such as sales territories, insurance rating territories, or any geographic areas that are of interest to you.	Optional	User-created
Sales Tax Cross-Reference Files Sales tax cross-reference files allow you to use the Enterprise Tax Module to determine tax jurisdictions for a given address, then use third-party software to determine the sales tax rates	Optional	Pitney Bowes Software

Database Name & Description	Required or Optional	Supplier
for those jurisdictions. The cross-reference files combine the U.S. Government Federal Information Processing Standards (FIPS) codes with the proprietary geocodes used by tax software from third parties. • **Vertex Files—**The Vertex files enable you to integrate the Enterprise Tax Module with tax compliance software from Vertex, Inc. With this file, the Enterprise Tax Module component AssignGeoTAXInfo can return the nine-digit Vertex jurisdiction code for an address. You can then match these codes to the Vertex tax tables, which tell you the tax rate for each jurisdiction. To use the Vertex file you must have either the MatchMaster file from Vertex or the PBBI Vertex file from Pitney Bowes Software to build the cross reference. • **Taxware Files—**The Taxware files enable you to integrate the Enterprise Tax Module with tax compliance software from ADP/Taxware. With this file, the Enterprise Tax Module component AssignGeoTAXInfo can return the Taxware jurisdiction code for an address. You can then match these codes to the Taxware tax tables, which tell you the tax rate for each jurisdiction. The Taxware files are supplied by Pitney Bowes Software and support both the SUT and TWE Taxware versions.		
Payroll Tax Cross-Reference Files Payroll tax cross-reference files allow you to use third-party software to determine the payroll tax rates. The cross-reference files combine the U.S. Government Federal Information Processing Standards (FIPS) codes with the proprietary geocodes used by tax software from third parties. The Payroll System Tax Code file is a customized file that you build to return the payroll tax codes used by your payroll system. For more information, see **AssignGeoTAXInfo** on page 651.	Optional	Pitney Bowes Software

AssignGeoTAXInfo

AssignGeoTAXInfo identifies the tax districts that apply to a given address. Specifically, AssignGeoTAXInfo returns information about an address:

- Latitude/longitude
- FIPS state codes and county codes

- County names
- MCD/CCD codes and names
- Place codes and names
- Incorporated or unincorporated status codes
- Cross-reference tax keys
- Result indicators
- Optionally, the relationship of an address to user-defined polygons

AssignGeoTAXInfo optionally includes enhanced tax jurisdiction information for an address, including:

- **Insurance premium districts**—Areas designated for the collection of taxes imposed on insurance policy premiums, based on the policy holder's address. Insurance premium districts are created by state governments.
- **Payroll tax districts**—Areas designated for the collection of taxes imposed on employers to support state or local government facilities and services, based on the employee's and/or employer's address. Examples include taxes collected for districts to pay for schools, police, or other services. Payroll tax districts are created by state or local governments.
- **Payroll system tax codes**—Codes that represent specific jurisdictions that collect payroll tax. Using payroll system tax codes has advantages over using the payroll tax district information returned by Assign GeoTAX Info:
- AssignGeoTAXInfo uses an additional database to determine payroll tax codes, resulting in more accurate payroll tax determination.
- Many payroll systems use specific codes to determine withholding amounts. Since you can customize the payroll tax codes returned by AssignGeoTAXInfo, you can set up a process where AssignGeoTAXInfo returns the exact payroll tax codes required by your payroll system, instead of returning jurisdictional IDs that must then be translated into the codes used by your system.

For more information, see **Using a Payroll Tax Correspondence File** on page 663.

- **Property tax districts**—Areas designated for the collection of taxes imposed on property owners to support local government facilities and services, based on the property's location. Examples include taxes collected for districts to pay for schools, police, or other services. Property tax districts are created by local governments.
- **Special purpose tax districts**—Areas designated for the collection of taxes imposed on residents to support specialized services for residents of the district, based on the resident's address. Examples include services such as sewer service, transit service, or water resources. Special purpose tax districts are created by legislative action, court action, or public referendums.

This optional information requires the use of boundary files which require an additional license. Contact your Pitney Bowes Software sales representative for more information.

AssignGeoTAXInfo is part of the Enterprise Tax Module. For a listing of other components, see **What Is the Enterprise Tax Module?** on page 648.

Determining Tax Rates with AssignGeoTAXInfo

AssignGeoTAXInfo does not determine tax rates, it determines tax jurisdictions that apply to a given location. You must use other software or a custom process to determine the tax rates for the jurisdictions returned by AssignGeoTAXInfo. There are two ways of determining tax rates: use software from Vertex or ADP/Taxware, or look up tax rates using your own custom process

- Using Vertex or ADP/Taxware Software
  a) Click the **Data** tab.

b) Under **Include data**, select the **Tax Jurisdiction** check box. This option returns several tax jurisdiction fields. The GeoTAXKey field contains the Vertex or Taxware code. For more information on these fields, see **Tax Jurisdiction** on page 685.

c) In the **Sales tax cross-reference key** field, select `Vertex MatchMaster` or `PBBI Vertex` to have AssignGeoTAXInfo return a Vertex key, or select `Taxware` to have AssignGeoTAXInfo return a Taxware key.

- Using a Custom Process

  If your organization has defined custom tax codes, you can use the jurisdiction information returned by AssignGeoTAXInfo to look up the tax jurisdictions in your custom tables. To do this, you need to configure AssignGeoTAXInfo to return the following fields:

  - StateCode
  - County.Code
  - GNISCode
  - SPDn.DistrictCode

  To include these fields in the output,

  a) Click the **Data** tab.

  b) Under **Include data**, select the **Census** check box. This option returns the **StateCode**, **County.Code**, and **GNISCode** fields that you will need to look up tax rates. For more information on these fields, see **Census** on page 667.

  c) In the `Tax district` field, select **Special Purpose District**. This option returns the SPDnDistrictCode field that you will need to look up tax rates, as well as several other fields that describe any special purpose districts that apply to the location. For more information, see **Special Purpose Tax District** on page 684.

Address Matching

AssignGeoTAXInfo takes a house address and matches it to the correct street segment. Using the house number, it determines the side of the street on which the house resides (usually based on an odd-even division). Knowing the correct side of the street is important because streets are often the boundaries between municipalities or other adjoining jurisdictions that may have different tax rates. Thus, two addresses on the same street, in the same city, within the same ZIP Code could exist in different jurisdictions and have different tax rates.

After AssignGeoTAXInfo knows the correct street segment, it determines the actual physical location of the house based on known latitudes/longitudes and other geographic data in the street segment database. AssignGeoTAXInfo returns the coordinates of the house, along with other data.

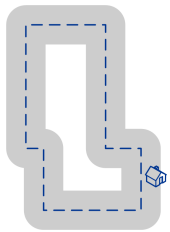
Note: Before using AssignGeoTAXInfo, you need to standardize your address data using a product such as the Universal Addressing Module.

Buffering

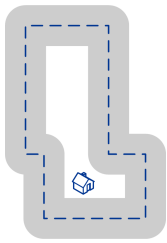
Use buffering to define areas that are close to the edges of a polygon, line, or point.

Buffered Polygon (zone)	Buffered Line (corridor)	Buffered Point (circle)

For example, if you work for an insurance company you may want to know if a potential customer's house is within 500 feet of a flood plain so that you can suggest that they buy flood insurance even though they are not actually within the flood plain. The following illustration shows this scenario using a buffered polygon. The dotted line indicates the boundary of the flood plain and the shaded area shows a 500-foot buffer zone around the boundary.

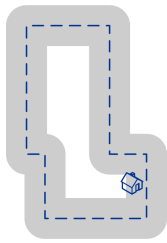


The buffer area extends on both sides of the boundary (inside and outside). When you use buffering, the output field BufferRelation indicates whether or not the point is in the buffered zone, and whether the point is inside or outside of the polygon, as shown in the following illustrations.



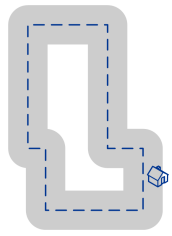
The point is inside the polygon and not in the buffer area.

The output field BufferRelation will contain "P".



The point is inside the polygon and in the buffer area.

The output field BufferRelation will contain "I".



The point is outside the polygon but in the buffer area.

The output field BufferRelation will contain "B".

Specify the size of polygon buffers using the BufferWidth input field to set it on a record-by-record basis and the DefaultBufferWidth option to set a default polygon buffer width for the job.

Input

AssignGeoTAXInfo takes a standard address data format as input. You should validate your address data by using ValidateAddress before using AssignGeoTAXInfo.

The following table provides information on the format and layout of AssignGeoTAXInfo input.

Note: Yspecify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 198: AssignGeoTAXInfo Input Data

columnName	Format	Description / Valid Values
AddressLine1	String [100]	First address line
AddressLine2	String [100]	Second address line
AddressLine3	String [100]	Third address line
AddressLine4	String [100]	Fourth address line
BufferWidth	String [10]	Specifies the width of the polygon buffers to use for Boundary File processing. The buffer width is used to determine if a point is close to the edge of a polygon. The output field BufferRelation indicates whether or not the point is within the polygon's buffer area. For more information, see **Buffering** on page 653. This field overrides the value specified in the DefaultBufferWidth option. If you do not specify a buffer width in this input field, the default is used. Specify the border width in the units specified by the DistanceUnits option.
City	String [50]	City name
Country	String [var]	The country where the address resides. The data you enter in this field has no impact on processing. It is simply passed through to output. **Note:** AssignGeoTAXInfo only supports US addresses.
FirmName	String [var]	Company or firm name
PostalCode	String [9]	Nine-digit ZIP Code

columnName	Format	Description / Valid Values
StateProvince	String [50]	The state where the address resides. The data you enter in this field has no impact on processing. It is simply passed through to output.
UserBufferWidth	Long [10]	Specifies the width of the polygon buffers to use for User-Defined Boundary File processing. The buffer width is used to determine if a point is close to the edge of a polygon. The output field BufferRelation indicates whether or not the point is within the polygon's buffer area. For more information, see **Buffering** on page 653. This field overrides the value specified in the DefaultUserBufferWidth option. If you do not specify a buffer width in this input field, the default is used. Specify the border width in the units specified by the DistanceUnits option.

Options

Set default AssignGeoTAXInfo processing options in Management Console. You can override certain settings for individual calls to AssignGeoTAXInfo using the Client API or Spectrum™ Technology Platform client tools, such as Enterprise Designer. If you do not specify system-level defaults through Management Console or a custom interface, and you do not pass any processing options at the time of the call, Spectrum™ Technology Platform uses the default options.

The processing options are:

- **Data** on page 656
- **Geocoding** on page 660
- **Output Format** on page 662

Data

Data options control the data returned by AssignGeoTAXInfo. **Table 199: AssignGeoTAXInfo Data Options** on page 656 lists the output data options.

Table 199: AssignGeoTAXInfo Data Options

optionName	Description/Valid Values
Database.GTX	The name of the database resource that contains the data to use in the search process. Use the database name specified in the Management Console's Database Resources tool. For more information, see the *Spectrum™ Technology Platform User's Guide*.

optionName	Description/Valid Values
UseStreetLevelMatching	Specifies whether or not AssignGeoTAXInfo should attempt a match to the street level data. One of the following: **Y** Yes, use the street matcher (default). **N** No, do not use the street matcher.
UseGeoTaxAuxiliaryFile	Specifies whether or not AssignGeoTAXInfo should attempt a match to the GeoTAX Auxiliary file. The GeoTAX Auxiliary file contains new addresses that have not yet been added to the Master File. Because matching to the GeoTAX Auxiliary file involves street-level matching, you must specify UseStreetLevelMatching=Y in order for this option to work. One of the following: **Y** Yes, attempt to match to GeoTAX Auxiliary file (default). **N** No, do not use the GeoTAX Auxiliary file.
UseAuxiliaryFile	Specifies whether or not AssignGeoTAXInfo should attempt a match to a User Auxiliary file. User Auxiliary files are user-defined files that the Enterprise Tax Module uses to override results from the master files in street-level matching. For more information, see **What Is the Enterprise Tax Module?** on page 648. Because matching to the User Auxiliary file involves street-level matching, you must specify UseStreetLevelMatching=Y in order for this option to work. One of the following: **Y** Yes, attempt to match to User Auxiliary file. **N** No, do not use the User Auxiliary file (default).
UseStateProvidedFile	Specifies whether or not AssignGeoTAXInfo should attempt a match to the state-supplied file. Use this option in combination with FileSearchOrder to specify a state-supplied file to use. State-supplied files are provided by individual state governments. By matching to the state-supplied files, you can remain compliant with tax jurisdiction assignment requirements mandated by new federal and state laws, such as the Mobile Telecommunications Sourcing Act and the Florida state Communications Services Tax Simplification Law. There are two supported file formats: the Florida-native format and the national TS-158 format (ANSI Transaction Set No. 158). The state of

optionName	Description/Valid Values
	Florida provides address files in both the TS-158 and its own native format. The state of Washington provides address data in the TS-158 format. Assign GeoTAX Info attempts to match to the state supplied file first. If a state match cannot be found, it attempts a match to the master files. You must install the appropriate state-supplied file to use these options. For instructions, see the *Spectrum™ Technology Platform Installation Guide*. One of the following: **Y** Yes, attempt to match to state-supplied file. **N** Do not use the state-supplied file (default).
FileSearchOrder	Specifies which state-supplied file to use. This option only takes effect if you specify UseStateProvidedFile=Y. One of the following: **FLOnly** Use only the Florida-native formatted file. **TSOnly** Use only the TS-158 formatted file.
UseRelaxedSecondaryMatching	Specifies whether or not AssignGeoTAXInfo matches input addresses with secondary information to records without secondary information. This option applies only to Florida-native files. One of the following: **Y** Yes, use relaxed matching. **N** No, do not use relaxed matching (default).
GeoTAXOutputRecordType	Select one or more of the following to obtain the type of data you want returned. AssignGeoTAXInfo groups the output fields into record types. If you do not want all of the fields in a record type returned, do not select the check box, and list only those fields you want returned in Extra Output Fields. • **C**—Census • **L**—Latitude/Longitude • **T**—Tax Jurisdiction • **U**—User-defined boundary file • **W**—Payroll System Tax Codes • **X**—Auxiliary File You can also specify one, and only one, of the following: **I** Insurance Premium Tax District (IPD)

optionName	Description/Valid Values
	P Property Tax District (PTD) **R** Payroll Tax District (PAY) **S** Special Purpose Tax District (SPD) For a description of the fields in each output group, see **Output** on page 666. **Note:** If you specify `W`, also specify `R` to obtain the best payroll system tax code match possible.
TaxKey	If you integrate AssignGeoTAXInfo with third-party tax compliance software from Vertex or ADP/Taxware, select which vendor you use. This controls the value returned in the GeoTAXKey output field. One of the following: **T** Return the Taxware jurisdiction code for the address. **Y** Return the Vertex jurisdiction code for the address. Select this option if you obtained a MatchMaster file from Vertex. **V** Return the Vertex jurisdiction code for the address. Select this option if you obtained a Vertex file from Pitney Bowes Software. **N** Do not return either the Taxware or Vertex jurisdiction codes (default).
OutputFields	Indicates the individual output fields you want returned. You can use this field instead of the Output Record Type to limit the output to those fields that are important to your current data needs. For a list of the fields included in each data type, see **Output** on page 666.

Specifying Default State-Supplied File Options

If you use the Spectrum™ Technology Platform API, note that the value you specify in the Management Console for the **State supplied file** field controls the default settings for three AssignGeoTAXInfo API options: UseStateProvidedFile, UseRelaxedSecondary, and FileSearchOrder. The following table show how each value in the **State supplied file** field affects these three options.

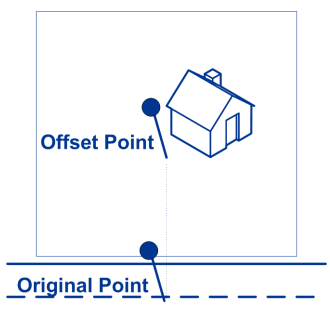
"State supplied file" value	UseStateProvidedFile	FileSearchOrder	UseRelaxedSecondary
None	N	N/A	N

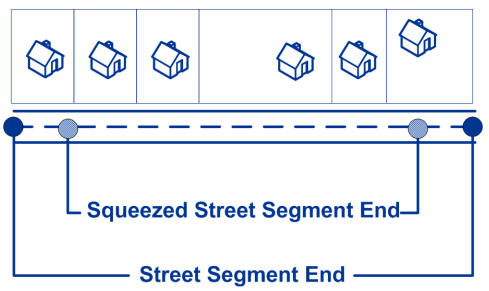
"State supplied file" value	UseStateProvidedFile	FileSearchOrder	UseRelaxedSecondary
Florida-native	Y	FSOnly	N
Florida-native with relaxed secondary matching	Y	FSOnly	Y
TS-158	Y	TSOnly	N

Geocoding

Geocoding is the process of determining the latitude/longitude coordinates of a given address. Address coordinates are used as the basis for determining the tax jurisdictions for an address. Geocoding options control how AssignGeoTAXInfo determines address latitude and longitude.

Table 200: AssignGeoTAXInfo Geocoding Options

optionName	Description/Valid Values
LatLongOffset	Indicates the offset distance in feet from the street center line. The offset distance is used in street-level geocoding to prevent the geocode from being in the middle of a street. It compensates for the fact that street-level geocoding returns a latitude and longitude point in the center of the street where the address is located. Since the building represented by an address is not on the street itself, you do not want the geocode for an address to be a point on the street. Instead, you want the geocode to represent the location of the building which sits next to the street. For example, an offset of 50 feet means that the geocode will represent a point 50 feet back from the center of the street. The distance is calculated perpendicular to the portion of the street segment for the address. Offset is also used to prevent addresses across the street from each other from being given the same point. The following diagram shows an offset point in relation to the original point. 

optionName	Description/Valid Values
	One of the following: **0** No offset (default). **20** Twenty feet offset from street center line **40** Forty feet offset from street center line **60** Sixty feet offset from street center line
Squeeze	Specifies if AssignGeoTAXInfo should squeeze the street end points when determining the geocode of an address in street-level matching. The squeeze value is 5% (2.5% from each endpoint). The following diagram compares the end points of a street segment to the squeezed end points of a street segment.  One of the following: **N** No, do not apply squeeze. **Y** Apply squeeze (default).
LatLongFormat	Indicates the desired format for returned latitude/longitude. Options include: **PreZero** Latitude/longitude in the following format: 090000000N180000000W. (Default) **PreZeroDecimal** Latitude/longitude in the following format: 090.000000N180.000000W. **Decimal** Latitude/longitude in the following format: 90.000000-180.000000. **DecimalAssumed** Latitude/longitude in the following format: 90000000-180000000. **DegMinSec** Latitude/longitude in the following format: 90 00 00N180 00 00W.

optionName	Description/Valid Values
DistanceUnits	Specifies the units in which to measure distance. One of the following: **Miles** Distances are measured in miles. **Km** Distances are measured in kilometers. **Feet** Distances are measured in feet. (Default) **Meters** Distances are measured in meters.
DefaultBufferWidth	Specifies the buffer width to use for tax district boundary files. The tax district boundary files are the Special Purpose District (SPD) file, the Insurance Premium District (IPD) file, the Payroll Tax District (PAY) file and the Personal Property Tax District (PTD) file. For more information on these files, see **What Is the Enterprise Tax Module?** on page 648. Specify the distance in the units of measurement specified in the **DistanceUnits** option. The default buffer width that you specify here can be overridden on a record-by-record basis using the BufferWidth input field. For more information on buffers, see **Buffering** on page 653.
DefaultUserBufferWidth	Specifies the buffer width to use for user-defined boundary files. Specify the distance in the units of measurement specified in the **DistanceUnits** option. For information on buffers, see **Buffering** on page 653. The default buffer width that you specify here can be overridden on a record-by-record basis using the BufferWidth input field. To use buffers, your user-defined boundary file must support buffers. For more information on user-defined boundary files, see **Creating a User-Defined Boundary File** on page 663.

Output Format

Output format options control how AssignGeoTAXInfo formats output data. The following table lists the output format options.

Table 201: AssignGeoTAXInfo Output Format Options

optionName	Description/Valid Values
OutputCasing	Specifies the casing of these output fields: County.Name, MSA.Name, MCD.Name, Place.Name, IPDn.DistrictName, PAYn.DistrictName, SPDn.DistrictName, and PTCn.PayrollDescription.

optionName	Description/Valid Values
	One of the following: **M** The output in mixed case (default). For example: Rensselaer. **U** The output in upper case. For example: RENSSELAER.

Creating a User-Defined Boundary File

User-defined boundary files define areas of interest to your organization, such as sales territories or insurance rating territories. AssignGeoTAXInfo uses this data to determine if an address falls within an area of interest. For example, you can create a boundary file that defines your sales territories. AssignGeoTAXInfo can then determine the tax jurisdictions that apply to an address and the sales territory of that address.

Note: User-defined boundary files in AssignGeoTAXInfo allow you to perform basic "point-in-polygon" spatial analysis. Point In Polygon provides additional point-in-polygon features.

1. Create an ESRI shapefile (.SHP) or MapInfo data interchange format file (.MIF) with the boundaries you want. Your .SHP or .MIF file can have up to three columns of user-defined data. The first and second columns are 10 bytes long and the third column is 50 bytes long. For information on .SHP and .MIF files, see **User-Defined Centrus Databases** on page 715.
2. Use the Boundary File Conversion utility to convert your .SHP or .MIF file to a .TXB file. This utility is available on the GeoTAX Utilities CD. This CD also contains instructions on using the conversion utility.

Using a Payroll Tax Correspondence File

Payroll system tax codes are proprietary codes used by some payroll tax applications to represent specific jurisdictions or combinations of jurisdictions. If you have licensed this option, you can use a payroll tax correspondence (PTC) file to determine the payroll system tax codes for a given address.

To use a payroll system tax code database, customize the PTC file and then install the file.

1. Customize the PTC file.

   After you receive your initial PTC file, modify it using a text editor of your choice. Specifically, you need to modify the following:

   - **Description**—A meaningful description of the code that represents business rules within your organization.
   - **Flags**—Flags indicate the payroll codes you want Assign GeoTAX Info to return. Possible flag values are:
   - **N**—No, do not return this payroll tax code. Continue searching the PTC file for other matching records. A blank flag has the same effect as "N".
   - **D**—Done. Return this payroll system tax code and stop searching the PTC file for other matching records.

- **Any other value**—Any other value indicates to return the code. Typically, the letter Y (for "yes") is used. Use other flags if you like. For example, if there is an area with two codes, one for a work location and the other for the worker's residence, you could use flags of "W" and "R" so that the output field PTCn.PayrollFlag returned by AssignGeoTAX Info indicates the type of location.

The following table shows how to use flags.

Note: Each record in the PTC file can contain up to six payroll system tax codes and their associated descriptions and flags. In the following example, the first three codes are on the first record and the second three are on the second. See **Table 203: Payroll System Tax Code File Layout** on page 665 for the layout of the PTC file.

Table 202: Example PTC File

Record	Description	Payrolll System Tax Code	Flag
1	HARBORCREEK TWP	123ABC	D
	HARBOR CREEK SD	456DEF	D
	HARBORCREEK TWP (M + SD)	789GHI	N
2	DAYTON BORO	592UID	Y
	ARMSTRONG SD	143XMA	Y
	DAYTON BORO (M + SD)	592JKT	N

In this example, the first record in the PTC file contains tax codes for a municipality called Harbor Creek Township and a school district called Harbor Creek School District. There is a separate code that represents points that are in both the Harbor Creek Township and the Harbor Creek School District. The second record in the PTC file contains codes for Dayton, a school district called Armstrong, and points located in both Dayton and the Armstrong School District.

This example returns the payroll system tax codes for the individual taxing jurisdictions for a given address, not the codes that represent the combined jurisdictions.

For addresses located in both the Harbor Creek Township and Harbor Creek School District, AssignGeoTAXInfo returns the following:

- PTC1.PayrollDescription=HARBORCREEK TWP
- PTC1.PayrollCode=123ABC
- PTC1.PayrollFlag=D
- PTC2.PayrollDescription=HARBOR CREEK SD
- PTC2.PayrollCode=456DEF
- PTC2.PayrollFlag=D

For addresses located in both Dayton and the Armstrong School District, AssignGeoTAXInfo returns the following:

- PTC1.PayrollDescription=DAYTON BORO
- PTC1.PayrollCode=592UID

- PTC1.PayrollFlag=Y
- PTC2.PayrollDescription=ARMSTRONG SD
- PTC2.PayrollCode=143XMA
- PTC2.PayrollFlag=Y
- For addresses located in just Dayton but not the Armstrong School District, AssignGeoTAXInfo returns the code 592UID and its associated flag and description. For addresses located in just the Armstrong School District but not Dayton, AssignGeoTAXInfo returns the code 143XMA and its associated flag and description.

Note: For a listing of output fields returned for payroll system tax codes, see **Payroll System Tax Code** on page 676.

The following table describes the PTC file layout.

Table 203: Payroll System Tax Code File Layout

Position	Length	Name
1	9	(required) Key Value. Lookup key into the file. Must be one of the following. • 5 character jurisdiction ID from the Pay.txb boundary file. This file contains polygons for special areas, such as school districts. • 9 character GNIS code. • 5 character county key made up of the State and County FIPS codes. • 2 character State FIPS code.
10	2	(optional) State
12	30	(optional) County
The payroll code and payroll description fields are paired. Each pair has a corresponding flag field. The flag field determines if GeoTAX returns data for the corresponding payroll code and description fields. GeoTAX does not return the fields if the flag is N or blank. This lets you create meaningful flags based on your business rules.		
42	15	Payroll code 1
57	40	Payroll description 1
97	15	Payroll code 2
112	40	Payroll description 2
152	15	Payroll code 3
167	40	Payroll description 3
207	15	Payroll code 4

Position	Length	Name
222	40	Payroll description 4
262	15	Payroll code 5
277	40	Payroll description 5
317	15	Payroll code 6
332	40	Payroll description 6
372	1	Flag 1
373	1	Flag 2
374	1	Flag 3
375	1	Flag 4
376	1	Flag 5
377	1	Flag 6

2. Install the PTC file. For instructions on installing the PTC file, see the *Spectrum™ Technology Platform Installation Guide*

PTC file updates are provided at intervals defined in your contract. When you receive the PTC update files, enter flags for the modified records and then run Enterprise Tax Module database load utility to merge the updated PTC file with your existing file. For instructions, see "Installing Payroll Tax Correspondence Files" in the *Spectrum™ Technology Platform Installation Guide*. This process maintains the existing user-defined flags you have already associated with the PTC data.

Output

AssignGeoTAXInfo output always contains these categories of output:

- **Input Address** on page 669
- **Result Indicators** on page 680

You can also include these optional categories of information:

- **Auxiliary File** on page 667
- **Census** on page 667
- **Insurance Premium Tax District** on page 670
- **Latitude/Longitude** on page 674
- **Payroll System Tax Code** on page 676

- **Payroll Tax District** on page 676
- **Property Tax District** on page 678
- **Special Purpose Tax District** on page 684
- **Tax Jurisdiction** on page 685
- **User-Defined Boundary File** on page 689

Note: If you are using the API to interact with this service, tThe output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Auxiliary File

The following table lists the output fields that contain auxiliary file data. To include auxiliary file data in the output, set **GeoTAXOutputRecordType** = X. The following table lists the output fields that contain tax jurisdiction data.

Table 204: Auxiliary File Output Fields

columnName	Description
AuxiliaryData.AuxiliaryFile	Data retrieved as a result of an auxiliary match from the user-defined area of the auxiliary file.
AuxiliaryData.StateFile	Data retrieved as a result of a state match. Data content and format vary depending on the state file used.

Census

The census output fields contains census information from the U.S. Census, including Minor Civil Divisions (MCDs) and Census County Division (CCD) names and codes. MCDs are the primary political or administrative divisions of a county, representing many kinds of legal entities with a variety of governmental and administrative functions. CCDs are established in states where there are no legally established MCDs. The Census Bureau recognizes MCDs in 28 states and has established CCDs in 21 states. The District of Columbia has no primary divisions, and the city of Washington, DC is considered equivalent to an MCD for data presentation purposes.

Census data also contains the Federal Information Processing Standards (FIPS) codes for each state and county. The FIPS State Code and the FIPS County Code are both used by the Census Bureau to identify these geographic units.

Table 205: Census Data Output Fields on page 668 lists the output fields that contain census data. To include census data in the output, set **GeoTAXOutputRecordType** = C.

Table 205: Census Data Output Fields

columnName	Description
Census.BlockCode	Census Block Group code.
Census.MatchCode	The level of match obtained against the databases. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output. **A** Auxiliary street match **G** State file address match **S** Street address match **U** GeoTAX Auxiliary file match **9** ZIP + 4 Code level match **5** ZIP Code level match **null** Unsuccessful match
Census.MatchLevel	The level of match obtained against the databases. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output. **Auxiliary** Auxiliary street match **Aux2** GeoTAX Auxiliary file match **Street** Street address match **Gov** State file address match **ZIP+4** ZIP + 4 Code level match **ZIP** ZIP Code level match **null** No match
Census.Tract	Six-digit tract number extracted from the Census.BlockCode.
County.Code	Three-digit Federal Information Processing Standards (FIPS) county code extracted from the Census.BlockCode. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output.

columnName	Description
County.Name	Name of the county. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output.
GNISCode	Unique nine-digit Geographic Names Information System (GNIS) code. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output.
MCD.Code	Minor Civil Division/Census County Division (MCD/CCD) Code.
MCD.Name	Minor Civil Division/Census County Division (MCD/CCD) name.
MSA.Code	Metropolitan Statistical area (MSA) code.
MSA.Name	Metropolitan Statistical area (MSA) name.
State.Abbreviation	Two-character state abbreviation. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output
StateCode	Two-digit Federal Information Processing Standards (FIPS) state code extracted from the Census.BlockCode. **Note:** This field is always included in the output regardless of whether or not you choose to include census data in the output.

Input Address

AssignGeoTAXInfo always returns the input address as part of the output. The input address fields are returned as input from the data. AssignGeoTAXInfo does not change these input values.

Table 206: Input Address Output Fields

columnName	Description
AddressLine1	Input address line 1.
AddressLine2	Input address line 2.

columnName	Description
AddressLine3	Input address line 3.
AddressLine4	Input address line 4.
City	Input address city.
Country	Input address country.
FirmName	Input address firm name.
PostalCode	Input address postal code
StateProvince	Input address state.

Insurance Premium Tax District

The following table lists the output fields that contain Insurance Premium Tax District (IPD) data. For more information on insurance premium tax districts, see **AssignGeoTAXInfo** on page 651. To include census data in the output, set GeoTAXOutputRecordType = I.

Note: AssignGeoTAXInfo returns multiple districts for IPDs, SPDs, PTDs, and PAYs.

Table 207: Insurance Premium Tax District Output Fields

columnName	Description
IPD*n*.BoundaryBuffer.BufferRelation	Indicates where in the district the address resides in relation to the edge of the district. One of the following: **P** The address is inside the district at a distance from the edge that is greater than the specified buffer width. Buffer width is specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **I** The address is inside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**.

columnName	Description
	B The address is outside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. For more information, see **Buffering** on page 653.
IPD*n*.BoundaryBuffer.DistanceToBorder	Indicates the distance from the address to the border of the district. The distance is in the units specified by the option DistanceUnits.
IPD*n*.DistrictID	IPD ID.
IPD*n*.DistrictName	IPD name.
IPD*n*.DistrictType	IPD district type.
IPD*n*.UpdateDate	IPD update date.
IPD*n*.VersionDate	IPD compiled date.
IPD*n*.Notes	Tax code descriptions. Field Length: 20 Possible Values: 01, 33, A, B
IPD*n*.ChangeDate	IPD change date.
IPD*n*.EffectiveDate	MMDDYY - Identifies when district becomes active - State supplied Field Length: 6 Possible Values: 010108
IPD*n*.ExpirationDate	MMDDYY - Identifies when district becomes inactive - State supplied Field Length: 6 Possible Values: 063009
IPD*n*.FireRate	Format is dependent on associated flag Field Length: 20

columnName	Description
	Possible Values: .13, 15.00 or 3;7
IPD*n*.FireFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semi colon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.CasualtyRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.CasualtyFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.VehicleRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.VehicleFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.MarineRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.MarineFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.HealthRate	Format is dependent on associated flag Field Length: 20

columnName	Description
	Possible Values: .13, 15.00 or 3;7
IPD*n*.HealthFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.LifeRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.LifeFlag	P - Percentage; .1 = 10%, .0575 = 5.75% P - Percentage; .1 = 10%, .0575 = 5.75% M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.OtherRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.OtherFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
IPD*n*.MinimumRate	Format is dependent on associated flag Field Length: 20 Possible Values: .13, 15.00 or 3;7
IPD*n*.MinimumFlag	P - Percentage; .1 = 10%, .0575 = 5.75% F - Flat Fee dollar amount M - Multiple Percentages has a semicolon as a delimiter. 3;7 = "3% or 7%"
NumberIPDsFound	Number of IPDs returned.

Latitude/Longitude

The following table lists the output fields that contain latitude and longitude data. Latitude/Longitude data contains the coordinates for the address and additional information about how AssignGeoTAXInfo determined the latitude and longitude. To include latitude/longitude data in the output, set **GeoTAXOutputRecordType** = L.

Table 208: Latitude/Longitude Output Fields

columnName	Description	
Latitude	Seven-digit number in degrees and calculated to four decimal places (in the format you specified).	
Latitude.Directional	Latitude directional.	
	N	North
	S	South
LatLong	Returned latitude/longitude, in the format you specified (up to 22 alphanumeric characters).	
LatLong.MatchCode	Level of match for the latitude/longitude.	
	2	ZIP + 2 centroid
	4	ZIP + 4 Code centroid
	5	ZIP Code centroid based on a ZIP + 4 code.
	B	Block group centroid
	R	Address-level based on street address
	T	Census tract centroid
	U	Address-level match using the GeoTAX Auxiliary Database
	Z	ZIP Code centroid based on a five-digit ZIP code
	null	No latitude/longitude determined
LatLong.MatchLevel	Level of match for the latitude/longitude.	
	Block	Block group centroid
	Rooftop	Exact address match
	Tract	Census tract centroid

columnName	Description	
	ZIP	ZIP Code centroid
	ZIP+2	ZIP + 2 centroid
	ZIP+4	ZIP + 4 centroid
	Auxiliary	Address-level match using the GeoTAX Auxiliary Database
LatLong.StreetMatchCode	Output street address return code.	
	H	House number not found on street
	L	Latitude/longitude not determined on auxiliary match
	S	Street not found in ZIP Code
	Z	ZIP Code not found in street address database
	N	Street-level matching option not selected
	null	The street was successfully matched
LatLong.StreetMatchLevel	Street level match used to determine the latitude/longitude.	
	FullMatch	Successful match
	HouseNotFound	House number not found on street
	LatLongNotFound	Latitude/longitude not determined on auxiliary match
	StreetNotFound	Street not found in ZIP Code
	ZipNotFound	ZIP Code not found in street address database
	NotUsed	Street-level matching option not selected
Longitude	Seven-digit number in degrees and calculated to four decimal places (in the format specified).	
Longitude.Directional	Longitude directional.	
	E	East
	W	West

Payroll System Tax Code

The following table lists the output fields that contain Payroll System Tax Code (PTC) data. For more information on payroll tax districts, see **AssignGeoTAXInfo** on page 651. To include this data in the output, set **GeoTAXOutputRecordType** = W.

Note: AssignGeoTAXInfo returns up to six payroll tax codes per address.

Table 209: Payroll System Tax Code Output Fields

columnName	Description
NumberPTCsFound	Number of payroll system tax codes found for this address.
PTC.MatchCode	Indicates the level of match obtained for the address. In order from most specific match to least, the possible match codes are: **P** The address was matched to a specific Payroll District ID. This is the most specific match. **G** The address was matched to a GNIS Code. **F** The address was matched to a county's FIPS code. **S** The address was matched to a state's FIPS code. This is the least specific match.
PTCn.PayrollCode	A code that represents a taxing authority in a payroll application. This is a user-defined code. The specific codes are determined by the payroll application that utilizes the data returned by AssignGeoTAXInfo.
PTCn.PayrollDescription	A description of the purpose of this payroll code.
PTCn.PayrollFlag	A user-defined flag from the PTC database.
StateCounty	The state abbreviation and county name.

Payroll Tax District

The following table lists the output fields that contain Payroll Tax District (PAY) data. For more information on payroll tax districts, see **AssignGeoTAXInfo** on page 651. To include this data in the output, set **GeoTAXOutputRecordType** = R.

Note: AssignGeoTAXInfo returns multiple districts for IPDs, SPDs, PTDs, and PAYs.

Table 210: Payroll Tax District Output Fields

columnName	Description
NumberPAYsFound	Number of PAYs returned.
PAYn.BoundaryBuffer.BufferRelation	Indicates where in the district the address resides in relation to the edge of the district. One of the following: **P** The address is inside the district at a distance from the edge that is greater than the specified buffer width. Buffer width is specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **I** The address is inside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **B** The address is outside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. For more information, see **Buffering** on page 653.
PAYn.BoundaryBuffer.DistanceToBorder	Indicates the distance from the address to the border of the district. The distance is in the units specified by the option DistanceUnits.
PAYn.DistrictID	PAY district ID.
PAYn.DistrictName	PAY district name.
PAYn.DistrictType	PAY district type.
PAYn.ID	PAY ID.
PAYn.MunicipalEMSTax	PAY municipality emergency municipal services tax. The values for Pennsylvania are: **Y** Levies the tax **N** Does not levy the tax All other states are null.

columnName	Description
PAYn.MunicipalIncomeTax	PAY municipality income tax. The values for Pennsylvania are: **R** Resident **N** Non-resident **B** Both **X** None All other states are null.
PAYn.SchoolDistrictEMSTax	PAY school district emergency municipal services tax. The Values for Pennsylvania are: **Y** Levies the tax **N** Does not levy the tax All other states are null.
PAYn.SchoolDistrictIncomeTax	PAY school district income tax. The values for Pennsylvania are: **R** Resident **N** Non-resident **B** Both **X** N The values for Ohio are: **R** Resident **X** None All other states are null.

Property Tax District

The following table lists the output fields that contain Property Tax District (PTD) data. For more information on property tax districts, see **AssignGeoTAXInfo** on page 651. To include this data in the output, set **GeoTAXOutputRecordType** = P.

Note: AssignGeoTAXInfo returns multiple districts for IPDs, SPDs, PTDs, and PAYs.

Table 211: Property Tax District Output Fields

columnName	Description
NumberPTDsFound	Number of PTDs returned.
PTDn.BoundaryBuffer.BufferRelation	Indicates where in the district the address resides in relation to the edge of the district. One of the following: **P** The address is inside the district at a distance from the edge that is greater than the specified buffer width. Buffer width is specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **I** The address is inside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **B** The address is outside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. For more information, see **Buffering** on page 653.
PTDn.BoundaryBuffer.DistanceToBorder	Indicates the distance from the address to the border of the district. The distance is in the units specified by the option DistanceUnits.
PTDn.DistrictID	PTD district ID.
PTDn.DistrictName	PTD district name.
PTDn.DistrictType	PTD district type. Only returned for Indiana. **R** Reporting district **B** Billing district
PTDn.JurisdictionID	PTD jurisdiction ID.
PTDn.UpdateDate	PTD update date.

Result Indicators

The following table lists the output fields that contain results data. Result indicators describe how well AssignGeoTAXInfo matched the input address to a known address and assigned a location. These fields are always included in output from AssignGeoTAXInfo.

Table 212: Result Indicator Output Fields

columnName	Description
Confidence	Indicates the confidence in the output provided; from 0 to 100. The higher the score, the higher the confidence in the match. Calculated based on the match results for individual output fields, using the following algorithm: Census.MatchCode + Latlong.StreetMatchCode + LatLong.MatchCode The maximum confidence score is 100, so if this calculation results in a value greater than 100, the Confidence score is returned as 100. AssignGeoTAXInfo uses the following values: • Census.MatchCode • A = 85 • G = 85 • S = 85 • U = 85 • 9 = 65 • 5 = 45 • null = 0 • LatLong.StreetMatchCode • H = 5 • S = 0 • L = 0 • Z = -10 • null = 10 • LatLong.MatchCode • 2 = 0 • 4 = 5 • 5 = -5 • B = 2 • R = 10 • T = -2 • U = 10 • Z = -5 • null = -10

columnName	Description
GTX.ErrorCode	This field contains a return code if the GeoTAX engine experiences an abnormal termination. **Note:** This field contains the same set of codes returned by the standalone GeoTAX software and is intended for users who have migrated from GeoTAX to Spectrum™ Technology Platform. The first character indicates the file (or set of files affected). **A** Auxiliary file **B** Base file **D** Boundary file **F** User defined boundary file **G** Street file **L** Logical I/O program abend **S** State supplied files **U** GeoTAX Auxiliary file **X** Street and state files The second position is one of the following: **E** Both Florida and TS158 state files are problematic **F** Expired database **I** Informational
GTX.ErrorDescription	If the GeoTAX engine experiences an abnormal termination, this field contains a text description of the reason. It is blank if GeoTAX terminated normally. The maximum length is 80. **Note:** This field contains the same set of descriptions returned by the standalone GeoTAX software and is intended for users who have migrated from GeoTAX to Spectrum™ Technology Platform. SI-"TS158 FILES NOT FOUND" SI-"TS158 FILES VINTAGE OR INCOMPLETE DB ERROR" SI-"STATE FILES NOT FOUND" SE-"STATE AND TS158 FILES NOT FOUND" SE-"STATE NOT FOUND AND TS158 VINTAGE ERROR" SI-"STATE FILES VINTAGE OR INCOMPLETE DB ERROR" SE-"STATE VINTAGE ERROR AND TS158 NOT FOUND" SE-"STATE AND TS158 FILES VINTAGE OR INCOMPLETE DB

columnName	Description
	ERROR" GI-"STREET FILES NOT FOUND" XI-"STREET AND TS158 FILES NOT FOUND" XI-"STREET NOT FOUND AND TS158 FILES VINTAGE ERROR" XI-"STREET AND STATE FILES NOT FOUND" XE-"STREET STATE AND TS158 FILES NOT FOUND" XE-"STREET AND STATE NOT FOUND AND TS158 VINTAGE ERROR" XI-"STREET NOT FOUND AND STATE VINTAGE ERROR" XE-"STREET AND TS158 NOT FOUND AND STATE VINTAGE ERROR" XE-"STREET NOT FOUND AND STATE AND TS158 VINTAGE ERROR" GI-"STREET FILES VINTAGE OR INCOMPLETE DB ERROR" XI-"STREET VINTAGE ERROR AND TS158 NOT FOUND" XI-"STREET AND TS158 FILES VINTAGE OR INCOMPLETE DB ERROR" XI-"STREET VINTAGE ERROR AND STATE NOT FOUND" XE-"STREET VINTAGE ERROR AND STATE AND TS158 NOT FOUND" XE-"STREET AND TS158 VINTAGE ERROR AND STATE NOT FOUND" XI-"STREET AND STATE FILES VINTAGE OR INCOMPLETE DB ERROR" XE-"STREET AND STATE VINTAGE ERROR AND TS158 NOT FOUND" XE-"STREET STATE AND TS158 VINTAGE ERROR" LF-"INVALID FUNCTION PASSED TO GTDBLIO : " AI-"GENIO ERROR: FILE = G1GTAUX , FUNC = , ST = " UI-"GENIO ERROR: FILE = G1GTAX2 , FUNC = , ST = " XF-"The (DB Vintage) database has expired!" XF-"The (SPD file Vintage) SPD File has expired!" DI- "UNABLE TO VALIDATE BOUNDARY LICENSE" DI- "UNABLE TO OPEN BOUNDARY FILE" DI- "BOUNDARY FILE NOT FOUND" FI- "UNABLE TO VALIDATE USER BOUNDARY LICENSE" FI- "UNABLE TO OPEN USER BND FILE" FI- "USER BND FILE NOT FOUND"
GTX.WarnCode	This field contains warning codes returned by the GeoTAX engine. It is blank if no warnings were issued. A value of `WN` indicates a database will expire next month.

columnName	Description
	Note: This field contains the same set of codes returned by the standalone GeoTAX software and is intended for users who have migrated from GeoTAX to Spectrum™ Technology Platform.
GTX.WarnDescription	A text description of any warnings returned by the GeoTAX engine. **Note:** This field contains the same set of descriptions returned by the standalone GeoTAX software and is intended for users who have migrated from GeoTAX to Spectrum™ Technology Platform.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure. Some examples of failures are your license expired or you did not select any output record types and fields for AssignGeoTAXInfo to return.
Status.Code	If AssignGeoTAXInfo could not process the address, this field will show the reason. Currently there is one possible value for this field: Invalid Address.
Status.Description	If AssignGeoTAXInfo could not process the address, this field will show a description of the failure. One of the following: TS158 FILES NOT FOUND TS158 FILES VINTAGE OR INCOMPLETE DB ERROR STATE FILES NOT FOUND STATE AND TS158 FILES NOT FOUND STATE NOT FOUND AND TS158 VINTAGE ERROR STATE FILES VINTAGE OR INCOMPLETE DB ERROR STATE VINTAGE ERROR AND TS158 NOT FOUND STATE AND TS158 FILES VINTAGE OR INCOMPLETE DB ERROR STREET FILES NOT FOUND STREET AND TS158 FILES NOT FOUND STREET NOT FOUND AND TS158 FILES VINTAGE ERROR STREET AND STATE FILES NOT FOUND STREET STATE AND TS158 FILES NOT FOUND STREET AND STATE NOT FOUND AND TS158 VINTAGE ERROR STREET NOT FOUND AND STATE VINTAGE ERROR STREET AND TS158 NOT FOUND AND STATE VINTAGE ERROR STREET NOT FOUND AND STATE AND TS158 VINTAGE ERROR STREET FILES VINTAGE OR INCOMPLETE DB ERROR STREET VINTAGE ERROR AND TS158 NOT FOUND STREET AND TS158 FILES VINTAGE OR INCOMPLETE DB ERROR

columnName	Description
	STREET VINTAGE ERROR AND STATE NOT FOUND STREET VINTAGE ERROR AND STATE AND TS158 NOT FOUND STREET AND TS158 VINTAGE ERROR AND STATE NOT FOUND STREET AND STATE FILES VINTAGE OR INCOMPLETE DB ERROR STREET AND STATE VINTAGE ERROR AND TS158 NOT FOUND STREET STATE AND TS158 VINTAGE ERROR INVALID FUNCTION PASSED TO GTDBLIO : GENIO ERROR: FILE = G1GTAUX , FUNC = , ST = GENIO ERROR: FILE = G1GTAX2 , FUNC = , ST = The (DB Vintage) database has expired! The (SPD file Vintage) SPD File has expired! UNABLE TO VALIDATE BOUNDARY LICENSE UNABLE TO OPEN BOUNDARY FILE BOUNDARY FILE NOT FOUND UNABLE TO VALIDATE USER BOUNDARY LICENSE UNABLE TO OPEN USER BND FILE USER BND FILE NOT FOUND

Special Purpose Tax District

The following table lists the output fields that contain Special Purpose Tax District (SPD) data. For more information on special purpose tax districts, see **AssignGeoTAXInfo** on page 651. To include this data in the output, set **GeoTAXOutputRecordType** = S.

Note: AssignGeoTAXInfo returns multiple districts for IPDs, SPDs, PTDs, and PAYs.

Table 213: Special Purpose Tax District Output Fields

columnName	Description
NumberSPDsFound	Number of SPDs returned.
SPDn.BoundaryBuffer.BufferRelation	Indicates where in the district the address resides in relation to the edge of the district. One of the following: **P** The address is inside the district at a distance from the edge that is greater than the specified buffer width. Buffer width is specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **I** The address is inside the district but is close to the edge. This indicates that the address is in the buffer

columnName	Description
	area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **B** The address is outside the district but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. For more information, see **Buffering** on page 653.
SPDn.BoundaryBuffer.DistanceToBorder	Indicates the distance from the address to the border of the district. The distance is in the units specified by the option DistanceUnits.
SPDn.CompiledDate	SPD compiled date.
SPDn.DistrictCode	3-digit district type code.
SPDn.DistrictName	SPD name.
SPDn.DistrictNumber	SPD district number.
SPDn.EffectiveDate	SPD effective date.
SPDn.UpdateDate	SPD update date.
SPDn.VersionDate	SPD version date.

Tax Jurisdiction

Tax jurisdiction data contains information about the "place" where the address resides. A "place" is a geographic area defined on the basis of population criteria that vary by state; or, an area recognized as significant because it is located in an incorporated municipality. Places are used to determine tax jurisdiction.

Table 214: Tax Jurisdiction Output Fields on page 686 lists the output fields that contain tax jurisdiction data. To include tax jurisdiction data in the output, set **GeoTAXOutputRecordType** = T. The following table lists the output fields that contain tax jurisdiction data.

Table 214: Tax Jurisdiction Output Fields

columnName	Description
County.PostalCode Confidence	Indicates if the ZIP Code resides in the county. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output. **N** ZIP Code not wholly contained in the county. **Y** ZIP Code wholly contained in the county. **null** Unknown if ZIP Code is wholly contained in the county or a match was not made at the ZIP Code level. **Note:** Only available for 5-digit ZIP Code matches, not for street address matches.
GeoTAXKey	The value in this field varies depending on the option you specified in the TaxKey option: If you specified `T`, GeoTAXKey contains the proprietary codes used in ADP/Taxware tax compliance software. You can use this code in your ADP/Taxware application to find out the tax rate for the jurisdiction. If you specified `Y` or `V`, GeoTAXKey contains the proprietary Vertex® jurisdiction code (comprised of a two-digit Vertex® state code, three-digit FIPS county code, and four-digit Vertex® city code). You can use this code in your Vertex® application to find out the tax rate for the jurisdiction.
GeoTAXKey.MatchCode	Return code denoting the level of match obtained against the Vertex or Taxware cross reference files. **E** Exact match using five fields: FIPS state code, FIPS county code, FIPS place code, ZIP Code, and FIPS place name. **P** Partial match using four fields: FIPS state code, FIPS county code, FIPS place code, and ZIP Code. **A** Alternate match using two fields: ZIP Code, FIPS place name. This return code is available only when using Vertex. **N** Record is default coded based on valid state code. This return code is available only when using Vertex. **null** No matching record found.
GeoTAXKey.MatchLevel	A description of the value returned in the GeoTAXKey.MatchCode field.

columnName	Description
	Exact Exact match. See description in GeoTAXKey.MatchCode. **Partial** Partial match. See description in GeoTAXKey.MatchCode. **Alternate** Alternate match. See description in GeoTAXKey.MatchCode. **DefaultCode** Record is default coded. See description in GeoTAXKey.MatchCode. **NoMatch** No matching record found.
Place.ClassCode	Place class code. Place class codes are used to determine the proper taxing jurisdictions **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.Code	An identifier for a specific place. A "place" is a geographic area defined on the basis of population criteria that vary by state. Or, an area recognized as significant because it is located in an incorporated municipality. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.IncorporatedFlag	Indicates whether the address is located in an incorporated or unincorporated place. A "place" is a geographic area defined on the basis of population criteria that vary by state. Or, an area recognized as significant because it is located in an incorporated municipality. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output. **Inc** Incorporated place code. **Uninc** Unincorporated place code. **Unknown** Incorporation status unknown.
Place.LastAnnexedDate	Last annexed date, in the format MM/YYYY, representing the month and year of the most recent boundary change or the most recent available boundary information.

columnName	Description
	Note: This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.LastUpdatedDate	Last updated date, in the format MM/YYYY, reflecting the month and year when Tele Atlas North America updated the database to reflect attribute (name change, FIPS change, etc.) or boundary edits to the Place. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.LastVerifiedDate	Last verified date, in the format MM/YYYY, representing the month and year that Tele Atlas North America verified municipality change information. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.Name	The name of the "place" where the address is located. A "place" is a geographic area defined on the basis of population criteria that vary by state. Or, an area recognized as significant because it is located in an incorporated municipality. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output.
Place.PostalCode Confidence	Indicates if the ZIP Code resides in the place. **Note:** This field is always included in the output regardless of whether or not you choose to include tax jurisdiction data in the output. **N** ZIP Code not wholly contained in the place. **Y** ZIP Code wholly contained in the place. **null** Unknown if the ZIP Code is wholly contained in the place or a match was not made at the ZIP Code level. Only available for five-digit ZIP Code matches, not for street address matches.

User-Defined Boundary File

The following table lists the output fields that contain data returned from user-defined boundary files. For more information on user-defined boundary files, see **What Is the Enterprise Tax Module?** on page 648. To include this data in the output, set **GeoTAXOutputRecordType** = U.

Note: AssignGeoTAXInfo can return up to 10 user-defined areas for each input address.

Table 215: Output Fields for User-Defined Boundary Files

columnName	Description
NumberUserBoundariesFound	Number of user-defined polygons returned.
UserBoundaryn.BoundaryDescription	A description of the polygon.
UserBoundaryn.BoundaryID	The ID of the polygon as specified in the user-defined boundary file.
UserBoundaryn.BufferRelation	Indicates where in the polygon the address resides in relation to the edge of the area. One of the following: **P** The address is inside the polygon at a distance from the edge that is greater than the specified buffer width. Buffer width is specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **I** The address is inside the polygon but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. **B** The address is outside the polygon but is close to the edge. This indicates that the address is in the buffer area specified either by the option DefaultBufferWidth or by the input field **BufferWidth**. For more information, see **Buffering** on page 653.
UserBoundaryn.DistanceToBorder	Indicates the distance from the address to the border of the polygon. The distance is in the units specified by the option DistanceUnits.
UserBoundaryn.SupplementalBoundaryID	A supplemental ID as specified in the user-defined boundary file.

Reports

To create the report, drag the **Assign GeoTAX Info Summary Report** icon to the canvas. You do not need to draw a connector to the report.

Assign GeoTAX Info Summary Report

The Assign GeoTAX Info Summary Report contains general information about the job, such as the settings, number of records processed, performance statistics, and the database used. It also contains detailed statistics about the results of the job. This report contains the following sections.

Job Summary

This section contains information about the software and databases used for the job.

- **Software Version**—The version of the underlying software used by AssignGeoTAXInfo. Note that this is not the same as the Spectrum™ Technology Platform version number.
- **Base Database Version**—The version of the Base portion of the Master File database. For information on the Master File, see **What Is the Enterprise Tax Module?** on page 648.
- **Advanced Database Version**—The version of the Advanced portion of the Master File database. For information on the Master File, see **What Is the Enterprise Tax Module?** on page 648.
- **Street Database Version**—The version of the Street portion of the Master File database. For information on the Master File, see **What Is the Enterprise Tax Module?** on page 648.
- **Cross Reference Database Version**—The version of the cross-reference database used in this job. Cross-reference databases are used to determine jurisdiction codes for use with third-party tax software. For information on the Cross Reference databases, see **What Is the Enterprise Tax Module?** on page 648.

Address Matching Summary

This section describes the input address match counts.

- **Total Match Attempts**—Address matches attempted in this job.
- **Total Records Matched**—Input addresses that were matched to known locations.
- **Total Unmatched Records**—Input addresses that could not be matched to known addresses.

Address Matching Levels

This section describes how precisely AssignGeoTAXInfo matched an input address to a location. The more accurately AssignGeoTAXInfo determines a street segment for an input address, the more accurately it determines tax jurisdictions for the address.

- **User-Defined Auxiliary File Matches**—Addresses matched to the user-defined auxiliary file. For information on a user-defined auxiliary file, see **What Is the Enterprise Tax Module?** on page 648.
- **GeoTAX Auxiliary File Matches**—Addresses matched to the GeoTAX Auxiliary file. For information on the GeoTAX Auxiliary file, see **What Is the Enterprise Tax Module?** on page 648.
- **State-Level Matches**—Addresses matched to a state-supplied files.
- **Street-Level Matches**—Addresses matched to a specific street segment.
- **ZIP-Level Matches**—Addresses matched to a 5-digit ZIP Code.

- **ZIP + 4-Level Matches**—Addresses matched to a ZIP + 4 code.

GeoTAX Key Matching

This section describes how accurately AssignGeoTAXInfo determined GeoTAX keys. A GeoTAX key is used in conjunction with software from a third party to determine tax rates. If you do not choose to return a GeoTAX key, this section is blank.

- **GeoTAX Cross Reference Match Attempts**—Addresses that AssignGeoTAXInfo attempted to match to an cross reference file. These files are used to determine codes for use with third party tax software.
- **Unsuccessful GeoTAX Cross Reference Matches**—Addresses that AssignGeoTAXInfo attempted to determine a code but was unable.
- **Successful GeoTAX Cross Reference Matches**—Addresses that AssignGeoTAXInfo determined a code using a cross reference file. Addresses included in this count have a value in the **GeoTAXKey** output field.
- **Exact Matches Achieved**—Addresses matched exactly to a GeoTAX key. For more information about GeoTAX key match levels, see the description of the GeoTAXKey.MatchCode output field under **Tax Jurisdiction** on page 685.
- **Partial Matches Achieved**—Addresses partially matched to a GeoTAX key. For more information about GeoTAX key match levels, see the description of the GeoTAXKey.MatchCode output field under **Tax Jurisdiction** on page 685.
- **Alternate Matches Achieved**—Addresses matched to a GeoTAX key using an alternate, less accurate, method. For more information about GeoTAX key match levels, see the description of the GeoTAXKey.MatchCode output field under **Tax Jurisdiction** on page 685.
- **Default Matches Achieved**—Addresses matched to a GeoTAX key only at the state level. For more information about GeoTAX key match levels, see the description of the GeoTAXKey.MatchCode output field under **Tax Jurisdiction** on page 685.

Latitude/Longitude Matching

This section describes how precisely AssignGeoTAXInfo determined latitude/longitude coordinates of an address. These counts are based on match codes. For more information, on match codes, see **Latitude/Longitude** on page 674.

- **Total Lat/Long Matches**—Addresses that AssignGeoTAXInfo determined latitude/longitude coordinates. Addresses included in this count have any value other than null in the **LatLong.MatchCode** output field.
- **Total Lat/Long Unmatched**—Addresses that AssignGeoTAXInfo was unable to determine latitude/longitude. Addresses included in this count have a value of null in the **LatLong.MatchCode** output field.
- **GeoTAX Auxiliary-level Matches**—Latitude/longitude determined using the GeoTAX Auxiliary File.
- **Address-Level Matches**—The latitude/longitude represents the actual location of the address. This is the most accurate type of geocode. Addresses included in this count have a value of R in the **LatLong.MatchCode** output field.
- **ZIP + 4-Level Matches**—The latitude/longitude represents the center of the ZIP + 4 code in which the address is located. Addresses included in this count have a value of 4 in the **LatLong.MatchCode** output field.

- **Census Block Group-Level Matches**—The latitude/longitude represents the center of the address's Census block group. Addresses included in this count have a value of B in the **LatLong.MatchCode** output field.
- **ZIP Sector-Level Matches**—The latitude/longitude represents the center of the address's ZIP + 2 code. Addresses included in this count have a value of 2 in the **LatLong.MatchCode** output field.
- **Census Tract-Level Matches**—The latitude/longitude represents the center of the address's Census tract. Addresses included in this count have a value of T in the **LatLong.MatchCode** output field.
- **ZIP Code-Level Matches**—The latitude/longitude represents the center of the ZIP Code in which the address is located. Addresses included in this count have a value of Z or 5 in the **LatLong.MatchCode** output field.

Census Matching

This section describes how precisely AssignGeoTAXInfo determined an address location within the statistical areas defined by the U.S. Census.

- **Census Tracts Determined**—Addresses that AssignGeoTAXInfo determined a census tract. Addresses included in this count have a value in the **Census.Tract** output field.
- **Census Tracts Not Determined**—Addresses that AssignGeoTAXInfo was unable to determine a census tract. Addresses included in this count have no value in the **Census.Tract** output field.
- **State Codes Determined**—Addresses that AssignGeoTAXInfo determined a state. Addresses included in this count have a value in the **StateCode** output field.
- **State Codes Not Determined**—Addresses that AssignGeoTAXInfo did not determine a state. Addresses included in this count have no value in the **StateCode** output field.
- **County Codes Determined**—Addresses that AssignGeoTAXInfo determined a county. Addresses included in this count have a value in the **County.Code** output field.
- **County Codes Not Determined**—Addresses that AssignGeoTAXInfo did not determine a county. Addresses included in this count have no value in the **County.Code** output field.
- **Census Block Group Determined**—Addresses that AssignGeoTAXInfo determined a block group. Addresses included in this count have a value in the **Census.BlockCode** output field.
- **Census Block Groups Not Determined**—Addresses that AssignGeoTAXInfo was unable to determine a block group Addresses included in this count have no value in the **Census.BlockCode** output field.
- **MSA Codes Determined**—Addresses that AssignGeoTAXInfo determined a metropolitan statistical area (MSA). Addresses included in this count have a value in the **MSA.Code** output field.
- **MSA Codes Not Determined**—Addresses that AssignGeoTAXInfo did not determine a metropolitan statistical area (MSA). Addresses included in this count have no value in the **MSA.Code** output field.
- **MCD/CCD Codes Determined**—Addresses that AssignGeoTAXInfo determined a Minor Civil Division/Census County Division code. Addresses included in this count have a value in the **MCD.Code** output field.
- **MCD/CCD Codes Not Determined**—Addresses that AssignGeoTAXInfo was unable to determine a Minor Civil Division/Census County Division code. Addresses included in this count have no value in the **MCD.Code** output field.

Tax Jurisdiction Matching

This section describes how accurately AssignGeoTAXInfo determined an address place. The place information is used to determine tax jurisdictions.

- **Place Codes Determined**—Addresses that AssignGeoTAXInfo determined a place code. Addresses included in this count have a value in the **Place.Code** output field.
- **Place Codes Not Determined**—Addresses that AssignGeoTAXInfo was unable to determine a place code. Addresses included in this count have no value in the **Place.Code** output field.
- **Place Names Determined**—Addresses that AssignGeoTAXInfo determined a place name. Addresses included in this count have a value in the **Place.Name** output field.
- **Place Class Codes Determined**—Addresses that AssignGeoTAXInfo determined a place code. Addresses included in this count have a value in the **Place.ClassCode** output field.
- **Incorporated Places**—Addresses that reside in an incorporated municipality. Addresses included in this count have a value of Inc in the **Place.IncorporatedFlag** output field.
- **Unincorporated Places**—Addresses that reside in a place that is not an incorporated municipality. Addresses included in this count have a value of Uninc in the **Place.IncorporatedFlag** output field.
- **Place Last Annexed Date Found**—Places that AssignGeoTAXInfo determined the last date of annexation. Addresses included in this count have a value in the **Place.LastAnnexedDate** output field.
- **Place Last Verified Date Found**—Places that AssignGeoTAXInfo determined the last date that the place data was verified by the data provider. Addresses included in this count have a value in the **Place.LastVerifiedDate** output field.
- **Place Last Updated Date Found**—Places that AssignGeoTAXInfo determined the date that the place data was last updated by the data provider. Addresses included in this count have a value in the **Place.LastUpdatedDate** output field.

Tax District Matching

This section describes the number of matches to specific types of tax districts. The specific type of tax districts displayed in this section vary based the district type selected in the **Tax district** field.

Note: If you specify a buffer width in the **Tax district buffer** field, a single location could be counted in more than one district due to buffering. For more information about buffering, see **Buffering** on page 653.

- **Successful Payroll Tax File Matches**—Addresses that AssignGeoTAXInfo determined a payroll tax code using the Payroll Tax District boundary file. For more information on Payroll Tax Districts, see **Payroll Tax District** on page 676.
- **Unsuccessful Payroll Tax File Matches**—Addresses that AssignGeoTAXInfo was unable to determine a payroll tax code using the Payroll Tax District boundary file. For more information on Payroll Tax Districts, see **Payroll Tax District** on page 676.
- **Locations in Special Purpose Districts**—Addresses that reside in at least one Special Purpose District. For more information on Special Purpose Districts, see **Special Purpose Tax District** on page 684.
- **Locations in 1 Special Purpose District**—Addresses that reside in a single Special Purpose District. Addresses included in this count have a value of 1 in the **NumberSPDsFound** output field.
- **Locations in 2 Special Purpose Districts**—Addresses that reside in two overlapping Special Purpose Districts. Addresses included in this count have a value of 2 in the **NumberSPDsFound** output field.
- **Locations in 3 or 4 Special Purpose Districts**—Addresses that reside in three or four overlapping Special Purpose Districts. Addresses included in this count have a value of 3 or 4 in the **NumberSPDsFound** output field.

- **Locations in 5+ Special Purpose Districts**—Addresses that reside in five or more overlapping Special Purpose Districts. Addresses included in this count have a value of 5 or greater in the **NumberSPDsFound** output field.
- **Locations in User-Defined Districts**—Addresses that reside in at least one user-defined district. For more information on user-defined districts, see **User-Defined Boundary File** on page 689.
- **Locations in 1 User District**—Addresses that reside in one user-defined district. Addresses included in this count have a value of 1 in the **NumberUserBoundariesFound** output field.
- **Locations in 2 User Districts**—Addresses that reside in two overlapping user-defined districts. Addresses included in this count have a value of 2 in the **NumberUserBoundariesFound** output field.
- **Locations in 3+ User Districts**—Addresses that reside in three or more overlapping user-defined districts. Addresses included in this count have a value of 3 in the **NumberUserBoundariesFound** output field.

CalculateDistance

CalculateDistance takes two sets of latitude/longitude coordinates as input, calculates the distance between the coordinates, and returns the distance between the two points.

CalculateDistance is part of the Enterprise Tax Module. For more information on the Enterprise Tax Module, including a listing of other components included with it, see **What Is the Enterprise Tax Module?** on page 648.

Input

CalculateDistance takes latitude and longitude information as input. The following table provides information on the format and layout of the input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 216: CalculateDistance Input Data

columnName	Description/Valid Values
FirstLatitude	Latitude of the first point for which you want distance returned.
FirstLatitude.Directional	First latitude directional. **N** North **S** South
FirstLongitude	Longitude of the first point for which you want distance returned.
FirstLongitude.Directional	First longitude directional.

columnName	Description/Valid Values	
	E	East
	W	West
SecondLatitude	Latitude of the second point for which you want distance returned.	
SecondLatitude.Directional	Second latitude directional.	
	N	North
	S	South
SecondLongitude	Longitude of the second point for which you want distance returned.	
SecondLongitude.Directional	Second longitude directional.	
	E	East
	W	West

Options

Set default CalculateDistance processing options in Management Console. You can override certain settings for individual calls to CalculateDistance using the Client API or Spectrum™ Technology Platform client tools, such as Enterprise Designer. If you do not specify system-level defaults through the Management Console or a custom interface, and you do not pass any processing options at the time of the call, Spectrum™ Technology Platform uses the default options listed in the following table.

Table 217: Output Data and Format Options

optionName	Description/Valid Values	
LatLongFormat	Indicates the format of the input latitude/longitude. The options are:	
	DegMinSec	For example 90 00 00N180 00 00W.
	PreZero	(090000000N180000000W). Default.
	PreZeroDecimal	(090.000000N180.000000W)
	Decimal	(90.000000-180.000000)
	DecimalAssumed	(90000000-180000000)
ReturnUnits	Indicates the measurement units returned for distance calculation:	

optionName	Description/Valid Values
	• Miles • Km • Feet • Meters

Output

CalculateDistance always returns the Confidence field to indicate the confidence in the output provided.

If CalculateDistance fails to process the data, it returns the fields Status, Status.Code, and Status.Descriptions. These fields provide information on why CalculateDistance failed to process the data. Some examples of failures are your license expired or you did not select any output record types and fields for CalculateDistance to return. The following table provides the record-level qualifiers and data outputs for CalculateDistance.

Table 218: CalculateDistance Output Fields

columnName	Description
Distance	Distance between the two input coordinates in the units of measurement that you specified.
Status	Reports the success or failure of the match attempt: **null** Success **F** Failure
Status.Code	Reason for failure or error. If Status = F, Status.Code = Failure.
Status.Description	Description of the problem. If Status = F, Status.Description = Unable to compute distance.

Creating a User-Defined Auxiliary File

To enter data in a user-defined auxiliary file, use your own editor or data entry program. Sort the records by ascending ZIP Code, street name, street type, directional, low house range, and descending high house range to minimize the time required to street-level match.

The auxiliary file has a block size of 8K, key length of 47 bytes, and record length of 800 bytes. The sample file `SEQAUX` illustrates the layout of the file.

The following table describes the layout of the auxiliary file. After you create a file with this layout, install the file using the Enterprise Tax Module database load utility. For instructions, see the *Spectrum™ Technology Platform Installation Guide*.

Table 219: User-Defined Auxiliary File Layout

Position	Field Name	Description	Length
Input Key Area (Must be Unique)			
1-5	G1GTAUX-ZIP-CODE	ZIP Code	5
6-35	G1GTAUX-STREETNAME	Street name	30
36-39	G1GTAUX-STREET-TYPE	Street type	4
40-41	G1GTAUX-PRE-DIR	Predirectional	2
42-43	G1GTAUX-POST-DIR	Postdirectional	2
44-47	G1GTAUX-SEQ	Sequence number, from 0001-9999	4
Output Area			
48-58	G1GTAUX-OUTPUTRANGE-FROM	First number in the house number range (right adjusted, blankfilled)	11
59-69	G1GTAUX-OUTPUTRANGE-TO	Last number in the house number range (right adjusted, blankfilled)	11
70	G1GTAUX-OUTPUTODD-EVEN	• O = Odd • E = Even	1
71	G1GTAUX-OUTPUTASC-DESC	• A = Ascending • D = Descending	1

Position	Field Name	Description	Length
72	Reserved		
73-74	G1GTAUX-STATE-CODE	FIPS State Code	2,0
75-77	G1GTAUX-COUNTYCODE	FIPS County Code	3,0
78-83	G1GTAUX-CENSUSTRACT	6-digit Census Tract number	6,0
84	G1GTAUX-BLOCKGROUP	1-digit Block Group	1
85-92	Reserved		8
93-94	G1GTAUX-STATE-ABBV	USPS state abbreviation	2
95-119	G1GTAUX-COUNTYNAME	County name	25
120-124	G1GTAUX-MCD-CODE	5-digit MCD Code	5,0
125-164	G1GTAUX-MCD-NAME	MCD Name	40
165-168	G1GTAUX-MSA-CODE	4-digit MSA Code	4,0
169-218	G1GTAUX-MSA-NAME	MSA Name	50
219-223	G1GTAUX-PLACE-CODE	5-digit Place Code	5,0
224-263	G1GTAUX-PLACENAME	Place Name	40
264-265	G1GTAUX-PLACECLASS-CODE	2-digit Place Class Code	2
266	G1GTAUX-PLACE-INCFLAG	• U - Unincorporated • I - Incorporated	1
267-273	G1GTAUX-PLACE-LASTANNEXED	Place last annexed date, in the format MM/YYYY	7
274-280	G1GTAUX-PLACE-LASTUPDATED	Place last updated date, in the format MM/YYYY	7

Position	Field Name	Description	Length
281-287	G1GTAUX-PLACE-LASTVERIFIED	Place last verified date, in the format MM/YYYY	7
288-296	G1GTAUX-PLACE-GNIS	GNIS Code	9
297-500	Reserved		204
501-800	G1GTAUX-AUX-AREA	User-defined data	300

15

GeoConfidence Module

In this section:

- What Is the GeoConfidence Module? 702
- GeoConfidenceSurface 703

What Is the GeoConfidence Module?

The GeoConfidence Module is used to determine the probability that an address or street intersection is within a given area. The module takes an address or intersection's location (determined by Geocode US Address), converts that location to a point, line, or polygon (depending on the precision of the match), then compares that shape with a database of known shapes to see if the two overlap, and the percentage overlap. For example, you could use the GeoConfidence Module to make decisions on a flood zone rating based on how much overlap there is between an address's location and the flood zone data. Anything greater than a 95% overlap with a 100-year flood zone may indicate that the address is in the flood zone. Conversely, anything less than 95% could cause your business process to send the address to exception processing that might include a manual review.

An address or intersection can be geocoded to a point, an address along a street segment (an array of street segment points), ZIP + 4 centroid, ZIP + 2 centroid, or ZIP Code centroid (polygons). You can use these shapes (points, lines, or polygons) to compare with other shapes to determine overlap, which can be used to determine a risk or probability.

Different geoconfidence polygons are generated depending on the GeoConfidence result returned by the Enterprise Geocoding Module. Refer to the Enterprise Geocoding Module documentation for more information about the GeoConfidence information returned by the Enterprise Geocoding Module.

The GeoConfidence Module supports U.S. locations only.

Note: GeoConfidence uses services provided by the Enterprise Geocoding and Location Intelligence modules.

GeoConfidence Components

GeoConfidence deploys three dataflows that you can modify in Enterprise Designer. Each dataflow consists of various components that were installed with the Enterprise Geocoding and Location Intelligence modules.

For information about each component in the installed dataflows, see the relevant component chapter in the *Spectrum™ Technology Platform User's Guide*.

The names of the dataflows are:

- **GeoConfidenceSurface** This is the dataflow that creates the geoconfidence surface that can be used for further analysis. The input is the GeoConfidence information that is returned from the Enterprise Geocoding Module. Currently, only the Geocode US Address stage can return this information.
- **CreatePointsConvexHull** This is a subflow that is used by the GeoConfidenceSurface template. You should not need to make any changes to this subflow.
- **FloodRiskAnalysis** This is an example dataflow.

GeoConfidence Databases

GeoConfidence uses the same databases as the Enterprise Geocoding and Location Intelligence modules.

For information about how to add an Enterprise Geocoding Module U.S. database resource, see the *Spectrum™ Technology Platform User's Guide*.

For information about how to add a Location Intelligence Module spatial database resource, see the *Spectrum™ Technology Platform User's Guide*.

In addition to these databases, the GeoConfidence Module includes a database of ZIP Code polygons. This is used by GeoConfidenceSurface.

GeoConfidenceSurface

Input

The input fields for GeoConfidenceSurface are the output fields returned by the GeoConfidence output category of the Enterprise Geocoding Module. These fields are described below.

columnName	Description
GeoConfidenceCode	The value returned in this field indicates which geoconfidence surface type has been returned. Possible values are: **INTERSECTION** A geocode point for the intersection of two streets. **ADDRESS** An array of street segment points representing the street segment where the address is located. **POINT** If the geocoder was able to match the address using point data, the point geometry where the address is located. **POSTAL1** A geocode point for the ZIP centroid. **POSTAL2** An array of points for all street segments in the ZIP + 2 in which the address is located. **POSTAL3** An array of points for street segments in the ZIP + 4 in which the address is located. **ERROR** An error has occurred.
StreetSegmentPoints	An array of latitude/longitude values that represent the street segment points.

columnName	Description
	Note: This field contains values only if the `GeoConfidenceCode` field returns a value of `ADDRESS`, `POSTAL2`, or `POSTAL3`.
GeoConfidenceCentroidLatitude	The latitude of the centroid of the geoconfidence polygon.
GeoConfidenceCentroidLongitude	The longitude of the centroid of the geoconfidence polygon.

Output

The GeoConfidenceSurface output field contains the geoconfidence polygon.

columnName	Description
Geometry	A geoconfidence polygon that represents the returned geometry.

Customizing the GeoConfidence Module

The GeoConfidence Module deploys three dataflow templates that you can modify in Enterprise Designer. Each dataflow consists of various components that were installed with the Enterprise Geocoding and Location Intelligence modules. For more information about dataflow templates, see the *Spectrum™ Technology Platform Templates Guide*.

The names of the dataflow templates are:

- **GeoConfidenceSurface** This is the template that creates the geoconfidence surface that can be used for further analysis. The input is the GeoConfidence information that is returned from the Enterprise Geocoding Module. Currently, only the Geocode US Address stage can return this information. To customize this template, you must at minimum specify the 5-digit ZIP Code spatial source in the ZIP stage (Query Spatial).
- **CreatePointsConvexHull** This is a subflow that is used by the GeoConfidenceSurface template. You should not need to make any changes to this subflow.
- **FloodRiskAnalysis** This is an example template. To customize this template, you must at minimum specify Flood spatial source in the Find Nearest stage.

16

Location Intelligence Module

In this section:

- What Is the Location Intelligence Module? 706
- Centrus Database Resources 706
- Named Connections 707
- Named Tables 709
- Named Maps 711
- Named Tiles 713
- User-Defined Centrus Databases 715

What Is the Location Intelligence Module?

The Location Intelligence Module provides spatial analysis that allows you to determine the relationship between a location and areas or points of interest. You can perform a variety of spatial calculations with the Location Intelligence Module, such as area, union, intersection, point in polygon, and more.

Location Intelligence Components

- **Closest Site**—Determines which sites are closest to a given location.
- **Find Nearest**—Locates the points of interest (POI) that are nearest to a given location.
- **Point In Polygon**—Determines whether or not a location resides in a given area.
- **Query Spatial Data**—Allows you to query a spatial database using spatial functions as well as standard SQL functions.
- **Read Spatial Data**—Allows you to access spatial data in a variety of commonly-used spatial data formats.
- **Write Spatial Data**—Allows you to insert spatial data into a table of commonly-used spatial data formats.
- **Spatial Calculator**—Performs a variety of calculations on spatial data, such as determining the area of a polygon, determining the distance between to points, or measuring the perimeter of an area, among other functions.
- **Spatial Union**—Returns a geometry object which represents the union of two input geometry objects.

Location Intelligence Module Databases

Location Intelligence Module databases define regions or points of interest. You choose the databases that contain the information you are interested in using in your spatial analysis. Pitney Bowes Software has an extensive library of databases available for use with the Location Intelligence Module. For a listing of the available databases, see **www.pbinsight.com/products/data**. The Location Intelligence Module works with databases in Centrus GSB format and MapInfo TAB format.

Centrus Database Resources

Centrus database resources can be used by the Location Intelligence Module. Centrus data is used by Closest Site and Point in Polygon.

Adding a Centrus Database Resource

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Centrus Database Resources**.
3. Click **Add** to install a new database or click **Modify** to change an existing database resource.

4. If you are adding a new database, enter a name for the database resource in the **Name** field. The name can be anything you choose. You cannot modify the name of an existing database resource. Changing the name of an existing database resource would cause any services or jobs that reference the database resource to fail.
5. In the **Path** field, enter the full path to the database file(s). Centrus database files have a .gsb extension. The file names for .gsb files must be in all lower case on Unix and Linux systems for them to be available to Spectrum™ Technology Platform.
6. In the **Cache Size** field, select the amount of memory to use to use to cache data. In general, the larger the cache the better the performance. The options are:
   - **None**—Do not cache data.
   - **Medium**—Use up to 5 MB of memory to cache data. Use this setting if there is a small amount of memory available for caching.
   - **Large**—Use up to 20 MB of memory to cache data. Use this setting if you are processing a large number of records.
   - **Huge**—Use up to 100 MB of memory to cache data. Use this setting if you are processing a large number of records.
7. Click **OK**. The list of databases includes this information:
   - **Name**—The name you have given the database. This is the name to use when referencing this database from a stage or service.
   - **Path**—The location of the database file.
   - **Cache size**—The amount of memory to use to cache data from this database.
   - **Feature Type**—Indicates the type of spatial data contained in the database (Points, Polygons, or Lines). Points databases can be used with the Closest Site and Point In Polygon stages. Polygon databases and Line databases can be used with Point In Polygon.
   - **Buffered**—Indicates whether or not the database supports buffering, which is a Point in Polygon feature that allows you to define areas that are close to the edge of a point, polygon, or line.

The database resource now appears in the list of available database resources in Closest Site and Point in Polygon.

Named Connections

Location Intelligence Module can read from databases and file servers, using your existing data repository as input to jobs. The Location Intelligence Module uses named entities that are stored in an internal repository, that can be called by name to access data stored in databases. This section explains how to add, view, and manage named connections, which must be configured prior to using certain sources.

There are two types of Named Connections:JDBC (Oracle, SQL Server, and PostGres/GIS) and FDO (SQLite)

Adding or Modifying a JDBC Named Connection

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Connections**.
3. Click **Add** or **Modify**. The **Connection Properties** dialog box will appear.

4. Enter a name for the resource in the **Connection Name** field. The name can be anything you choose. You cannot modify the name of an existing database resource. Changing the name of an existing database resource would cause any services or jobs that reference the database resource to fail.
5. In the **Connection type** drop-down, select JDBC.
6. In the **URL** field, specify the JDBC connection URL to the database. This URL is in the format jdbc:[subprotocol]:[node]/[databaseName]. For example:

   - Oracle— `jdbc:oracle:thin:@pbs-lim:8080:pbspatial`
   - SQL Server—`jdbc:sqlserver://PBS-LIM:8080;databaseName=pbspatial`
   - PostGres/GIS— `jdbc:postgresql://111.111.11.111:8080/pbspatial`

   For connections to a SQL Server or PostGres/GIS instance, not all connection string options that would normally be available via a particular JDBC driver are supported. Since GeoTools is used to connect to both SQL Server and PostGres/GIS, only the described connection URLs are supported.
7. In the **User name** field, specify the user for connecting to the database resource.
8. In the **Password** field, specify the password for the user connecting to the database resource.
9. In the **Properties** field, specify any database specific connection properties. Consult the database documentation for property names and value types that can be appended to a connection string. For example, property value `networkProtocol` and key `tcp`.

   Note: Oracle is currently the only database to support properties from the dialog. For the PostGres/GIS and SQL Server append properties to the URL.

10. Click **Test** to make sure your connection is valid.
11. Click **OK**.

The Named Connection to the database resource now appears in the list of available resources in Find Nearest, Query Spatial Data, and Read Spatial Data. These connections can also be used to access Named Tables in database resources.

Adding or Modifying a FDO Named Connection

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Connections**.
3. Click **Add** or **Modify**. The **Connection Properties** dialog box will appear.
4. Enter a name for the resource in the **Connection Name** field. The name can be anything you choose. You cannot modify the name of an existing database resource. Changing the name of an existing database resource would cause any services or jobs that reference the database resource to fail.
5. In the **Connection type** drop-down, select FDO.
6. In the **Provider** field, enter `OSGeo.SQLite` as the data provider type.
7. In the **Properties** field, define a **Key** named `File` and the value for this key should point to the SQLite database. For example: `c:\data\my.sqlite`.
8. In the **Properties** field, specify any additional database specific connection properties required for your database connection. Consult the database documentation for property names and value types that can be appended to a connection string.
9. Click **Test** to make sure your connection is valid.
10. Click **OK**.

The Named Connection to the database resource now appears in the list of available resources in Find Nearest, Query Spatial Data, and Read Spatial Data. These connections can also be used to access Named Tables in database resources.

Deleting a Named Connection

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Connections**.
3. Highlight the named connection you want to remove and click **Delete**. You will receive a confirmation message asking if you wish to delete the named connection.
4. Click **OK** to confirm.

Named Tables

Location Intelligence Module stages, services, and operations require Named Tables to access various types of data. Named Tables are a resource that point to data, and can be used to access MapInfo TAB (including View tables), Shape, JDBC database files (includes XY tables), or FDO database files (including text tables). This procedure describes how to add or modify Location Intelligence Module Named Table resources. These named tables are stored in the Location Intelligence Module repository, and can be used by all spatial services.

If you haven't already done so, install the data on your system. For adding Named Tables that are in a database resource, you must first add the Named Connection to the database resource. There are two types of Named Connections. For information on adding Named Connections, see **Adding or Modifying a JDBC Named Connection** on page 707 or **Adding or Modifying a FDO Named Connection** on page 708.

Note: Only a user with `read`, `add`, and `modify` permissions have the ability to add Named Tables using the Management Console. For more information on assigning permissions, see the "Managing Security" section in the **Spectrum Spatial Administration Guide**.

Adding or Modifying a Named Table

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Tables**.
3. Click **Add** or **Modify**. The **Add Named Table** or **Update Named Table** dialog box will appear.
4. If you are adding a new Named Table, enter the repository path and name for the named table resource in the **Named table name** field. You must prepend the name of the named table with where in the repository the named table is going to be created. For example, if you are creating the `World` named table and are creating this named table in the `NamedTables` folder in the repository, the name you would enter in the **Named table name** field would be `/NamedTables/World`. You cannot modify the name of an existing named table resource. Changing the name of an existing resource would cause any services or jobs that reference the resource to fail.
5. In the **Named table type** field select one of the following:
   - Tab—Choose this option if the data is in MapInfo TAB format on the file system.
   - Shape—Choose this option if the spatial data is in ESRI Shape file format.

- Database—Choose this option if the data is in a JDBC database, and have created a Named Connection.
- FDO—Choose this option if the data is in a FDO (SQLite) database, and have created a Named Connection.
- View—Choose this option if the data is in MapInfo TAB format, and are using a SQL statement to display a particular view of the table.

Note: When modifying a Named Table that has not been created using the Management Console (i.e., manually created and uploaded using the Named Resource Service or WebDav) the resource type might display as `unknown`. In this instance, the database connection information (JDBC or FDO) for the table has not been stored in the Management Console. To resolve the connection information, create a Named Connection in the Mangement Console to match the connection information in the Named Table you are trying to modify, then apply the new connection to the Named Table.

6. If you selected Tab as the source type, specify the path to the TAB file in the **Path** field.
7. If you selected Shape File as the source type, specify the path to the shape file in the **Path** field.
8. If you selected Database as the source type, complete these fields:
   - Connection name—The named connection to the database that defines a set of parameters used to access the database. Select a name from the list to specify an existing named connection.
   - Schema—The database schema object. This is typically the same as the user name.
   - Table—The table in the selected database containing the data you want to read into the dataflow.
   - XY—Check this box if your data is an XY table. Specify the X and Y columns and the coordinate system for these columns. In addition, select the styling of this table by either selecting **None**, defining a **Style column**, or defining a **Named style**. If selecting a named style, you must define an existing named style in the repository, and include the entire path from the root, e.g., /NamedStyles/PointStylePin.
9. If you selected FDO as the source type, complete these fields:
   - Connection name—The named connection to the database that defines a set of parameters used to access the database. Select a name from the list to specify an existing named connection.
   - Schema—The database schema object. For FDO this should be `Default`.
   - Table—The table in the selected database containing the data you want to read into the dataflow.
10. If you selected View as the source type, specify the SQL query in the **SQL** field. A View named table provides the ability to create a layer in a map based on the query in the named table at runtime. Using a View, you can create a named view containing a filtered view from multiple resources, based on a SQL query. Since your SQL query contains a named table (e.g., select Country,Capital, Obj from "/NamedTables/WorldcapTable" where Country='INDIA') you can use views on any datasource, as the named table might have a datasource type as TAB, XY, Oracle, SQL Server, etc.

    Once your SQL query is complete, click **ValidateQuery** to confirm your SQL statement is valid.
11. Click **OK**.

The named table resource is now available for use in all spatial services, and in the list of available resources in Find Nearest, Query Spatial Data, and Read Spatial Data.

Deleting a Named Table

1. Open the Management Console.

2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Table**.
3. Highlight the named table you want to remove and click **Delete**. You will receive a confirmation message asking if you wish to delete the named table.
4. Click **OK** to confirm.

Named Maps

Location Intelligence Module stages, services, and operations use Named Maps to display various types of data (tables, layers, styles, etc.) in map form. Named Tables make up the base of a named map, and these act as layers displaying various types of information.

This procedure describes how to add or modify Location Intelligence Module Named Map resources. These named maps are stored in the Location Intelligence Module repository, and can be used by all spatial services. The Named Map tool takes an existing MWS file (MapInfo Workspace) which defines a map created in MapInfo Professional, and generates a named map in the repository. This tool assumes that you have already created the named tables needed for the named map.

If you haven't already done so, have the data accessible on your system. For adding Named Tables see **Adding or Modifying a Named Table** on page 709. For tables that are are in a database resource, you must first add the Named Connection to the database resource before creating the named table. There are two types of Named Connections. For information on adding Named Connections, see **Adding or Modifying a JDBC Named Connection** on page 707 or **Adding or Modifying a FDO Named Connection** on page 708.

Note: Only a user with `read`, `add`, and `modify` permissions have the ability to add resources using the Management Console. For more information on assigning permissions, see the "Managing Security" section in the **Spectrum Spatial Administration Guide**.

Adding or Modifying a Named Map

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Maps**.
3. Click **Add** or **Modify**. The **Add Named Map** or **Modify Named Map** dialog box will appear.
4. If you are adding a new Named Map, select the MWS file in the **MWS source** field using the browser button. This MWS file will be used to define the named map. This tool can either create new named tables, create inline tables, or use existing named tables for the named map. If you are using existing named tables for the various layer definitions in the MWS file, you must create the named tables prior to using this tool to create the named map. For instruction on how to create named tables see **Adding or Modifying a Named Table** on page 709.
5. In the **Named Map name** field enter the name for the named map to be created. You can either use the repository browser button and select the directory in the repository and enter the named map name, or enter the repository path and named map name information in the field. For example, if you are entering the information in the field, creating the `World` named map and are creating this named map in a `NamedMaps` folder in the repository, the name you would enter in the **Named map name** field would be `/NamedMaps/World`.
6. Select one of the three methods to generate the table information for the named map:

- **Use existing named table** option will populate the tables in the map using already existing named tables in the repository. The list of named tables will be populated in the list and you must verify that each of the named tables are valid in the named map (see next step).
- **Use inline tables** option will generate a name map with table definitions inline. This means that no named tables will be used, rather the named map will contain all relevant table information. The **Local data path** and the **Server data path** are used to define the new path from the referenced table files on the machine where the MWS file was created (local path), to the new table files located on the hosting machine (server path). The simplest way is to only specify the **Server data path**. This will replace all data paths in the MWS file (up to the name of the table) with the new path specified. This includes both relative or full paths. Defining the **Local data path** and the **Server data path** will replace any path (full or partial) containing the local data path with the server data path. If both local and server path are not defined, the named map is created and contains the same paths, for the tables, that are already defined in MWS file.
- **Create named tables** option will generate new named tables in the repository in the specified folder define in the **Named tables path** field. If the named table already exists in that folder, it will use the existing one, unless you have selected the **Overwrite named tables** option. In this case, all named tables will be overwritten with new named table definitions. When creating named tables, the data source name is used from the MWS file, and not the name of the TAB file. If the data source name does not exist, only then will the TAB file name be used as the name for the named table. The **Local data path** and the **Server data path** are used to define the new path from the referenced table files on the machine where the MWS file was created (local path), to the new table files located on the hosting machine (server path). The simplest way is to only specify the **Server data path**. This will replace all data paths in the MWS file (up to the name of the table) with the new path specified. This includes both relative or full paths. Defining the **Local data path** and the **Server data path** will replace any path (full or partial) containing the local data path with the server data path. If both local and server path are not defined, the named map is created and contains the same paths, for the tables, that are already defined in MWS file.

7. If you are adding a new named map with the **Use existing named table** option, or modifying a named map, then you must verify the named tables. The list of named tables will be populated in the list. You must verify that each of the named tables are valid in the named map. The tool searches the repository for named tables that use the same data as defined in the MWS file. All named tables must be valid before you can create the named map. Based on the Status symbol perform the following actions:

- —A named table exists in the repository and will be used for the named map. If you would like to change the named table being used for this data, from the drop down, select **Browse for a named table** and choose a new named table.
- —There are multiple named tables in the repository that use data specified in the MWS file. From the drop down select one of the valid named tables found in the repository. If you would like to change the named table being used for this data, from the drop down, select **Browse for a named table** and choose a new named table.

- —No named tables were found in the repository that match the data used in the MWS file. Either from the drop down, select **Browse for a named table** and choose a named table to be used for this data, or exit this process and create a new named table for the required data for the MWS file then restart the process of creating this named map.

8. Click **OK**.

The named map resource is now available for use in all spatial services.

Deleting a Named Map

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Map**.
3. Highlight the named map you want to remove and click **Delete**. You will receive a confirmation message asking if you wish to delete the named map.
4. Click **OK** to confirm.

Named Tiles

Location Intelligence Module Map Tiling Service uses named tiles to render maps. The named tiles are stored in the repository located in the `NamedTiles` directory under the root repository. For example, a default installation will place the named tile definition files at http://localhost:8080/RepositoryService/repository/default/NamedTiles. Named tiles reference a named map that is used for the map display of the tile.

For adding named tiles, you must already have a named map defined. You can create or modify named maps using the Named Map tool in the Management Console, or manually creating the named map and uploading it to the repository. For information on adding Named Maps, see **Adding or Modifying a Named Map** on page 711.

Note: Only a user with `read`, `add`, and `modify` permissions have the ability to add resources using the Management Console. For more information on assigning permissions, see the "Managing Security" section in the **Spectrum Spatial Administration Guide**.

Adding or Modifying a Named Tile

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Tiles**.
3. Click **Add** or **Modify**. The **Add Named Tile** or **Update Named Tile** dialog box will appear.
4. If you are adding a new Named Tile, enter the name of the new named tile including the repository path and name for the named tile resource in the **Name** field. You must prepend the name of the named tile with where in the repository the named tile is going to be created. For example, if you are creating the `WorldTile` named tile and are creating this named tile in the `NamedTiles` folder in

the repository, the name you would enter in the **Name** field would be `/NamedTiles/WorldTile`. You cannot modify the name of an existing named tile resource. Changing the name of an existing resource would cause any services or jobs that reference the resource to fail.

5. In the **Description** field add a metadata description of the named map. This information is presented to the user when a `getDescription` call is sent to the Map Tiling Service.
6. In the **Named map** field enter the name of the named map to be used as the source of the name tile. Either enter the name of the named map including the repository path of the resource in the repository, or use the repository browser button and select the named map from the repository.
7. In the **Coordinate system** field enter the coordinate system projection to host the named map. The Map Tiling Service will transform the named map into the projection defined. The projection is defined using the EPSG format. Either enter the coordinate system EPSG code, or use the coordinate system browser button and select a supported coordinate system.
8. In the **Minimum level** field enter the minimum zoom level to host the map. Must be greater than zero.

   The level, in combination with the tile width, tile height, and bounds of the named map, determines the zoom level of the tiles to be returned. The level shows how close the map image is to the Earth. Level 1 is the furthest away and is composed of one tile that the entire map will be drawn into. Level 2 is composed of 4 tiles, 2 across and 2 down. Each tile is a quarter of the entire map. Level 3 is composed of 16 tiles, 4 across and 4 down, and so on. The higher the level specified, the closer to the Earth the map image appears. For example, levels 1 to 3 usually show global or hemispheric detail, levels 4 to 15 show county/state/province level of detail and some larger cities, levels greater that 15 show street level views.

   What the levels actually display is dependant on the named map hosted by the Map Tile Service. For example, a level of 1 for a world map would be zoomed out to show the entire world. However, if you have a named map of only a city (for example, New York, Toronto, or London), a level of 1 would be zoomed out to show a street level map.
9. In the **Maximum level** field enter the maximum zoom level to host the map.
10. In the **Width** field enter the width of the tiles in pixels. Must be greater than or equal to 16 and must be a number that can be calculated from a 2^n equation (for example 2^4=16, 2^5=32, 2^8=256).
11. In the **Bounds** fields enter the north, east, south, and west bounds of the tile. These are oordinates in the coordinate system specified by the **Coordinate system** field.
12. In the **Mime types** field enter the available image types of tiles generated by the tile server for the named map. This is a comma separated list of image mime types. For example: image/png,image/jpeg,image/gif
13. In the **Expiry date** field enter the date on which the client should delete the tile from the cache, and request a new copy from the server. The **ExpiryDate** value must be specified in a W3C formatted date string as YYYY-MM-DD. The expiry date has effect only when using external caching software such as Squid or Apache.
14. In the **Resolution** field enter the resolution of the tile images in dots per inch (the number of individual dots that can be placed within the span of one linear inch). If not specified, the GlobalMapResolution for the MapTiling Service is used. The minimum dpi you can define is 72, anything less would render a poor quality image. If a value less than 72 is defined, the service will throw an exception.
15. In the **Map rendering** field select the rendering quality (anti-alias) of the tile images generated for non raster maps.

    - Speed—Choose this option if you desire quicker return of map tiles. The image quality will not be as high, as the anti-aliasing will be turned off.

- Quality—Choose this option if you desire higher quality map tiles. Anti-aliasing is turned on, so there will be improvements to image rendering, however there will be a time increase for tiles to be returned due to the additional image processing.

16. In the **Raster rendering** field select the rendering quality (anti-alias) of the tile images generated for raster maps.
    - Speed—Choose this option if you desire quicker return of map tiles. The image quality will not be as high, as the anti-aliasing will be turned off.
    - Quality—Choose this option if you desire higher quality map tiles. Anti-aliasing is turned on, so there will be improvements to image rendering, however there will be a time increase for tiles to be returned due to the additional image processing.
17. In the **Render labels** field select if you would like the service to render LabelLayers when generating a tile.
    - True—Choose this option if you would like labels on your tiles for label layers in this map
    - False—Choose this option if you do not want labels rendered on your tiles for label layers in this map.
18. In the **Pad factor** field enter the number of additional tiles around rendered tile to generate. The pad factor is used to prevent the clipping of labels when a label crosses a tile boundary. The **PadFactor** controls the amount of space is rendered around the requested tile with 0 meaning no padding, 1 meaning padding of 1 tile around the requested tile and so on. If a **PadFactor**preference is not specified, the global preference in the configuration will be used. If neither is specified, the default is 1.
19. In the **Background opacity** field enter the level of background opacity (transparency) for the tile images. The background opacity is defined on a scale from 0.0 (zero) to 1.0. Where 0.0 is completely transparent and 1.0 is completely opaque. For example a value of 0.75, would be 75 percent transparent.
20. Click **OK**.

The named tile resource is now available for use in the MapTiling Service

Deleting a Named Tile

1. Open the Management Console.
2. Expand **Modules** > **Location Intelligence** > **Tools** then click **Named Tiles**.
3. Highlight the named tile you want to remove and click **Delete**. You will receive a confirmation message asking if you wish to delete the named tile.
4. Click **OK** to confirm.

User-Defined Centrus Databases

Spectrum™ Technology Platform modules that perform spatial analysis depend on databases that define the polygons (or areas) and points that are of interest. The kinds of areas that these files can define is virtually unlimited; some examples are insurance rating territories, sales territories, and store locations. You can create you own custom databases that identify areas or points of interest specific to your organization.

To use a custom Cenrtus spatial database you first create a data file using a supported format, then use an import utility to convert the data file into a format usable by Spectrum™ Technology Platform. The following table summarizes the formats and utilities used to create a user-defined spatial databases.

Table 220: Summary of User-Defined Spatial Database Usage

Spectrum™ Technology Platform Module	Components That Support User-Defined Databases	Supported Format the for User-Defined Data	Import Utility
Enterprise Tax Module	AssignGeoTAXInfo	ESRI Shape (.SHP) MapInfo MIF (.MIF)	createtxb.exe
Location Intelligence Module	PointInPolygon	ESRI (.SHP) Comma delimited (.CSV) DBF format (.DBF) Microsoft Access (.MDB) Tab file (.TAB) Text (.TXT)	FMTMakr2.exe SpatialImport.exe

Creating a User-Defined Centrus Spatial Database

To create a user-defined Centrus spatial database, choose a format that is supported by the module you want to use to perform a spatial analysis. The following formats are supported by one or more Spectrum™ Technology Platform modules. See **User-Defined Centrus Databases** on page 715 for a listing of the specific formats that each module supports.

ESRI Shapefile

An ESRI shapefile (.SHP) can contain points, lines, and polygons. For more information on the shapefile format, see:

http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

MapInfo MIF/MID

The MIF/MID format is comprised of two files, one with an extension of MIF, the other with the same name, but the extension MID. The MIF file contains the actual points that represent the objects, while the MID file contains the text information about the objects. For a complete description of the format, see:

http://resource.mapinfo.com/static/files/document/1074660800077/interchange_file.pdf

MIF files contain a tokenized header, followed by a DATA section. The comma delimiter is required. If a `DELIMITER` token is present in the file, the file is not imported.

The next part of the file header contains the definitions of the columns in the MIF file. Feature names, as well as numeric data about the features, are stored here. One of these columns is used as the identifier to store with the object.

The column definitions begin with the COLUMN token, followed by the number of columns in the MIF file. Following the COLUMN token is a single line entry for each column. The first entry on the line is the column name, the second is the column type. The following example indicates three columns in the MIF file:

```
COLUMN 3 FEAT_NAME char (15) POPULATION integer AREA decimal (8,4)
```

Any column type can be used as the name, but char is the most common, since that represents a string. The number in parentheses following char indicates the largest string (which cannot exceed 127 characters). For a complete listing of column types, contact MapInfo.

The token DATA indicates that the header is complete, and the listing of objects follows. For each object, a number of attributes may be listed, such as PEN and BRUSH. These settings are ignored. MapInfo supports a variety of object types, though only a few are used to represent geographical objects. Three object types are supported—Region, Polyline, and Point.

In a MIF file, latitude/longitude coordinates must be represented in decimal degrees, with up to six decimals of precision. Longitude is the x coordinate, and latitude is the y coordinate. Western longitudes and southern latitudes are always negated. This means that for the U.S., longitudes are negated, while latitudes are positive. User coordinates are entered as whole numbers, as the decimal portion is ignored.

Region Definition

A region consists of one or more polygons. Each polygon is uniquely defined. In a MIF/MID file, you do not need to "close" a polygon. The last point in the polygon automatically attaches to the first point in a polygon with a line segment. The format for a region object is as follows:

```
REGION #pgons numpts1 x1 y1 x2 y2 . . numpts2 x1 y1 x2 y2 . .
```

Polyline Definition

A polyline (PLINE) is a line that contains any number of points to define its shape. The number of points in the line follows the PLINE identifier. Below is the format for a PLINE object.

```
PLINE numpts x1 y1 x2 y2 . .
```

Point Definition

A point is defined as a single coordinate pair. Below is the format for a point object.

```
POINT x y
```

MID files list the data, such as name, for objects defined in the MIF file. There is one entry in the MID file for each object in the MIF file. The entry in the MID file must be in the same order as the objects appear in the MIF file. The data in the MID file is delimited by commas, unless otherwise configured by the DELIMITER statement in the MIF file. The comma is the only supported delimiter.

As an example, recall the column definitions used earlier in the MIF file:

```
COLUMN 3 FEAT_NAME char (15) POPULATION integer AREA decimal (8,4)
```

With this column definition, and assuming comma delimiters, sample entries in the MID file appear as:

```
First Feature,180232,1000.2345 California,4345105,56023.3569 Last
Feature,12643,136.4000
```

Note: Columns are separated by a single comma. There is no header or other information contained in the MIF file.

MapInfo TAB

The MapInfo .tab file format is used by MapInfo Professional. For more information, see the *MapInfo Professional User Guide*.

DBF

The DBF format is a common database format. Spectrum™ Technology Platform only supports point data in DBF format. If you want to create a user-defined spatial file of lines or polygons, you must use the Shapefile format.

The DBF file must contain longitude and latitude fields using the following names:

- The longitude field must be named one of the following:
- LON
- LONGITUDE
- N_LON
- The latitude field must be named one of the following:
- LAT
- LATITUDE
- N_LAT

Note: The field types must be Character, Date, Numeric, or Float.

Microsoft Access Database

Database files created with Microsoft Access can be used with Spectrum™ Technology Platform if they can contain point data only. User-defined spatial file of lines or polygons must use the Shapefile format.

The .MDB file must contain fields using the following names:

- The longitude field must be named one of the following:
- LON
- LONGITUDE
- N_LON
- The latitude field must be named one of the following:
- LAT
- LATITUDE
- N_LAT

The file may also include other fields. Field names must not be longer than 10 characters.

Note: The field types must be Character, Date, Numeric, or Float.

Text/CSV

Text files (.TXT) and comma-delimited files (.CSV) can contain points only, not polygons. If you want to create a user-defined spatial file of lines or polygons, you must use the Shapefile format.

- The longitude field must be named one of the following:
- LON

- LONGITUDE
- N_LON
- The latitude field must be named one of the following:
- LAT
- LATITUDE
- N_LAT

The file may also include other fields. Field names must not be longer than 10 characters.

Importing a User-Defined Centrus Spatial Database

After you have created a custom Centrus spatial file in one of these formats, convert it to a format that is usable by the Spectrum™ Technology Platform module using the conversion utility supplied with the module. Each module has its own utility. All the fields in the input file are converted in the attribute file.

Note: These utilities work on Windows only.

Enterprise Tax Module Conversion Utility

The Enterprise Tax Module utility is `createtxb.exe`. It is located on the GeoTAX Utilities CD. The CD Also contains a PDF file with instructions on how to use the utility.

Location Intelligence Module Conversion Utility

The Location Intelligence Module utility has two conversion utilities: `FMTMakr2.exe` and `SpatialImport.exe`. These are located on the Spectrum™ Technology Platform server in:

```
<SpectrumInstallLocation>\modules\lim\lib
```

These utilities are both command line utilities. You may need to use one or both utilities to convert your data, depending on the format.

Converting .TXT and .CSV Files

Converting .TXT and .CSV files is a two-step process:

First, create a format file (.FMT) for your .TXT or .CSV file using the Data Formatter utility (FMTMAKR2.exe). This utility can be found on the Spectrum™ Technology Platform server in:

```
<SpectrumInstallLocation>\modules\lim\lib
```

In this location there is also a help file (FMTMAKR2.HLP) that contains complete instructions on using the Data Formatter utility.

Note: Field names are limited to 10 characters.

Second, use the SpatialImport.exe utility to import the data into a format usable by Spectrum™ Technology Platform. Follow the instructions in **Converting .SHP, TAB, and .DBF Files** on page 720 to convert the .TXT or .CSV file. Note that in order for SpatialImport.exe to work with .TXT or .CSV files, the .FMT file created by the Data Formatter utility must be present in the same directory as the .TXT or .CSV file.

Converting .SHP, TAB, and .DBF Files

If you are importing an ESRI Shapefile (.SHP), a tab file (.TAB), or a DBF file (.DBF) for use with the Location Intelligence Module, open a command prompt and type the following:

```
SpatialImport [-u] InputFile OutputFile [PolygonNameField]
```

Where:

- `-u` creates an unbuffered file. An unbuffered file will not allow you to specify buffers to use when matching. For more information on buffers, see the documentation on the component you are using. Using this option will create a much smaller .gsb file. Optional.
- `InputFile` is your user-defined file.
- `OutputFile` is the name of the .gsb file to create.

Note: On Unix and Linux systems, the name of the .gsb file must be in all lower case.

Note: Field names are limited to 10 characters.

- `PolygonNameField` is an optional parameter that specifies the field in the input file to use as the polygon name. This is the field that is used to populate the Polygon.Name output field returned by Spectrum™ Technology Platform. For example, if the fields in the user-defined file are LON, LAT, Congressman, URL, and PartyAffiliation, and you are using PointInPolygon, when PointInPolygon returns data for a match, it would return the Congressman, URL, and PartyAffiliation as attributes. If you were to specify Congressman in the `PolygonNameField` parameter, the value in the Congressman field will also be returned in the Polygon.Name output field.

Converting .MDB Files

If you are importing a Microsoft Access (.MDB) file for use with the Location Intelligence Module, type the following:

```
SpatialImport dbfile.mdb:tablename
```

Where:

- `dbfile.mdb` is the Access database that contains your user-defined spatial data.
- `tablename` is the table in the access database that contains your user-defined spatial data.

Note: Field names are limited to 10 characters.

17

Universal Addressing Module

In this section:

- What Is the Universal Addressing Module? 722
- AutoCompleteLoqate 734
- GetCandidateAddresses 740
- GetCandidateAddressesLoqate 750
- GetCityStateProvince 756
- GetCityStateProvinceLoqate 759
- GetPostalCodes 762
- GetPostalCodes Loqate 765
- ValidateAddress 767
- ValidateAddressAUS 834
- ValidateAddressGlobal 840
- ValidateAddressLoqate 861
- Encountering False Positives 886
- ValidateAddress Confidence Algorithm 889

What Is the Universal Addressing Module?

The Universal Addressing Module is an address quality module that can standardize and validate addresses, improving the deliverability of mail. The Universal Addressing Module can ensure that your address data adheres to quality standards established by the postal authority. An address that adheres to these standards is more likely to be delivered in a timely manner. In addition, mailers who follow these standards can qualify for significant postage discounts. For information on discounts for U.S. mail, refer to the USPS *Domestic Mail Manual* (*DMM*) available at **www.usps.com**. For information on discounts for Canadian mail, refer to the Canada Post website at **www.canadapost.ca**. For information on discounts for Australian mail, refer to the Australia Post website at **www.auspost.com.au**.

The Universal Addressing Module can be used in batch mode, realtime mode, or as a hosted service, depending on which option you have licensed. The batch version of the Universal Addressing Module is CASS Certified™ by the USPS®. It is also AMAS certified by Australia Post.

The Universal Addressing Module is one of two address quality modules available for Spectrum™ Technology Platform. The other address quality module, the Address Now Module, provides enhanced support for addresses outside the U.S. and Canada, including validation for more countries and double-byte support. If you have a large amount of international address data, you may want to consider using the Address Now Module for address standardization and validation.

Universal Addressing Components

The Universal Addressing Module consists of the following components. These components can work with U.S., Canadian, Australian, and international addresses as long as you are licensed for the appropriate database (if your are running Universal Addressing in your own environment) or hosted service (if you are utilizing Universal Addressing through the Pitney Bowes Software hosted services).

- **GetCandidateAddresses**—Returns a list of possible matches for a given address.
- **GetCityStateProvince**—Returns the city and state/province for a given postal code.
- **GetPostalCodes**—Returns the postal codes for a given city.
- **ValidateAddress**—Standardizes and validates addresses using U.S., Canadian, and international postal data.
- **ValidateAddressAUS**—Standardizes and validates addresses using Australian postal data.
- **ValidateAddressGlobal**—ValidateAddressGlobal provides enhanced address standardization and validation for addresses outside the U.S. and Canada. ValidateAddressGlobal can also validate addresses in the U.S. and Canada but its strength is validation of addresses in other countries. If you process a significant number of addresses outside the U.S. and Canada, you should consider using ValidateAddressGlobal.

Universal Addressing Databases

The Universal Addressing Module uses a number of required and optional databases. The databases are installed on the Spectrum™ Technology Platform server. Some of the databases are available by subscription from Pitney Bowes Software and are updated monthly or quarterly. Others are licensed from the USPS®. The following table lists the Universal Addressing databases.

Table 221: Universal Addressing Module Databases

Database Name & Description	Required or Optional	Supplier
U.S. Postal Database The U.S. Postal Database is in a Pitney Bowes proprietary format. It contains every house number range in the United States and is updated on a monthly basis. The database files contain the following information: • ZIP + 4® Code • Standardized address elements • City and state information The U.S. Postal Database also contains the data needed to perform Enhanced Street Matching (ESM) and All Street Matching (ASM). ESM and ASM apply extra matching logic to any input address that is not matched through the regular address validation process.	Required for U.S. address processing	Pitney Bowes Software monthly subscription
Canadian Postal Database The Canadian Postal database is in Pitney Bowes Software proprietary format. The database files contain the following information: • Postal code • Standardized address elements • Municipality and province information	Required for Canadian address processing	Pitney Bowes Software monthly subscription
Australia Post Postal Address File Database The Postal Address File is part of Australia Post's Address Matching Approval System (AMAS) program. The database file contains the following information: • Postal code • Standardized address elements	Required for Australian address processing	Pitney Bowes Software monthly subscription
International Postal Database The International Postal Database is a collection of postal address data from around the world. Data from each country is categorized according to the level of data available. The categories are:	Required for International address processing	Pitney Bowes Software quarterly subscription

Database Name & Description	Required or Optional	Supplier
• **Category A**—Enables the validation and correction of an address's postal code, city name, state/county name, street address elements, and country name. • **Category B**—Enables the validation and correction of an address's postal code, city name, state/county name, and country name. It does not support the validation or correction of street address elements. • **Category C**—Enables the validation and correction of the country name, and the validation of the format of the postal code.		
DPV® Database The Delivery Point Validation database allows you to check the validity of an individual mailing address in the U.S. (over 145 million locations). The DPV database enhances the U.S. Postal database's ability to validate mailing addresses. For more information on DPV, see **Delivery Point Validation (DPV)** on page 728. **Note:** The DPV database also contains the data required for Commercial Mail Receiving Agency (CMRA) processing. Each time an edition of the U.S. Postal database is released, a corresponding edition of the DPV database is released. Although USPS licensing allows the use of the U.S. Postal database beyond the expiration date (with certain restrictions), DPV lookups may not be performed after the expiration date of the DPV database. USPS licensing prohibits using DPV data for the generation of addresses or address lists. To prevent the generation of address lists, the DPV database contains "false positive records." False positive records are artificially manufactured addresses. For each negative response that occurs in a DPV query, a query is made to the False/Positive table in the DPV database. A match to this table will stop DPV processing. For more information, see **Encountering False Positives** on page 886. USPS licensing also prohibits exporting the DPV data outside the United States.	Optional, but required for CASS Certified™ processing; U.S. addresses only	Pitney Bowes Software monthly subscription

Database Name & Description	Required or Optional	Supplier
eLOT® Database The Enhanced Line of Travel (eLOT) database is a U.S. address database that ensures that Enhanced Carrier Route mailings are sorted as close as possible to the actual delivery sequence. the eLOT database is required for certain types of postal discounts. For more information, see **Enhanced Line of Travel (eLot)** on page 729. You will receive monthly updates to your eLOT database on the same media as the U.S. Postal database. You must install the U.S. Postal database and eLOT database from the same month (i.e., September eLOT data must be processed with a September U.S. Postal database). If the U.S. Postal database and the eLOT database are not from the same month, there may be ZIP + 4® Codes for which eLOT numbers cannot be assigned. The ZIP Code™, ZIP + 4 Code, carrier route code, and the delivery point of an address must be provided to assign a eLOT code.	Optional; U.S. addresses only	Pitney Bowes Software monthly subscription
EWS Database The Early Warning System (EWS) database prevents address validation errors that can result due to a delay in postal data reaching the U.S. Postal database. For more information, see **Early Warning System (EWS)** on page 729. The EWS database consists of partial address information limited to the ZIP Code™, street name, pre- and post-directionals, and a suffix. For an address record to be EWS-eligible, it must be an address not present on the most recent monthly production U.S. Postal database. The USPS® refreshes the EWS file on a weekly basis (Thursdays). You can download the EWS file from the USPS® website at **www.ribbs.usps.gov**.	Optional; U.S. addresses only	Download for free from USPS® website
LACS^{Link}® Database The LACS^{Link} database allows you to correct addresses that have changed as a result of a rural route address converting to street-style address, a PO Box renumbering, or a street-style address changing. For more information, see **Locatable Address Conversion System (LACS)** on page 729.	Optional, but required for CASS Certified™ processing; U.S. addresses only	Pitney Bowes Software monthly subscription

Database Name & Description	Required or Optional	Supplier
USPS licensing prohibits using LACS^{Link} for the generation of addresses or address lists. To prevent the generation of address lists, the LACS^{Link} database contains "false positive records." False positive records are artificially manufactured addresses. For each negative response that occurs in a LACS^{Link} query, a query is made to the False/Positive table in the LACS^{Link} database. A match to this table will stop LACS^{Link} processing. For more information, see **Encountering False Positives** on page 886. USPS licensing also prohibits exporting the LACS^{Link} database outside the United States		
RDI™ Database The Residential Delivery Indicator (RDI™) database contains data that can help you determine the best cost for shipping your packages. For more information, see **Residential Delivery Indicator (RDI)** on page 729. RDI is similar to DPV in that the RDI data is supplied as hash tables. However, RDI is a much simpler process than DPV in that the standard hash algorithm is only determined for the 9-digit and 11-digit ZIP Code™ rather than the entire address.	Optional; U.S. addresses only	License directly from USPS®
Suite$^{Link™}$ Database Suite$^{Link™}$ corrects secondary address information for U.S. business addresses whose secondary address information could not be validated. If Suite^{Link} processing is enabled, ValidateAddress attempts to match the value in the FirmName field to a database of known firm names. ValidateAddress then supplies the correct secondary address information. For more information, see **SuiteLink** on page 730.	Optional; U.S. addresses only	Pitney Bowes Software monthly subscription

U.S. Addressing Concepts

A U.S. address can be a street address, a rural route address, a highway contract address, a P.O. box address, or a private mailbox address.

A street address contains these elements:

4235	**N**	**MAIN**	**ST**	**NW**	**APT**	**100**

House Number	Leading Directional	Street Name	Suffix	Trailing Directional	Apartment Label	Apartment Number

A rural route, highway contract, or P.O. box address has these elements:

RR	**123**	**BOX**	**19**
Rural Route or Highway Contract Type	Rural Route or Highway Contract Number		Rural Route, Highway Contract, or PO Box Number

A private mailbox address contains these elements:

PMB	**345**
Private Mailbox Type	Private Mailbox Number

Address elements and city/state/ZIP Code information combine to make address lines. These lines are defined as shown in the figure below:

For complete information on U.S. address standards, see the U.S. Postal Service® (USPS®) publication *Postal Addressing Standards, USPS Publication 28*, available on the USPS website.

Coding Accuracy Support System (CASS)

The Coding Accuracy Support System (CASS™) is a United States Postal Service® (USPS®) program that certifies the accuracy of address validation software. To qualify for certain postal discounts you must use software that is CASS Certified™ to assign ZIP Code™, ZIP + 4® codes, and delivery point barcodes to mail.

Makers of address validation software must pass a test designed by the USPS in order to have their software designated as CASS Certified™. CASS Certified™ software must pass tests of accuracy in the following areas:

- Five-digit coding
- ZIP + 4®/delivery point (DP) coding
- Carrier route coding
- Delivery Point Validation (DPV®)
- Locatable Address Conversion System (LACS^{Link}®)
- Enhanced Line of Travel (eLOT®)
- Residential Delivery Indicator (RDI™)

When you use a CASS Certified™ product, you are assured of the following minimum levels of accuracy:

Table 222: CASS Certification Levels

Certification Level	Required Accuracy Level
ZIP	98.5%
Carrier Route	98.5%
ZIP+4	98.5%
Delivery Point Barcode	100%
eLOT	100%
Perfect Addresses	100%

Commercial Mail Receiving Agency (CMRA)

A Commercial Mail Receiving Agency (CMRA) is a private company that rents out mailboxes. A CMRA, also known as a mail drop, typically operates as a Private Mail Box Operator. Addresses at a CMRA are therefore given the designation "PMB" (private mail box) instead of "POB" (Post Office box).

A customer of a CMRA can receive mail and other deliveries at the street address of the CMRA rather than the customer's own street address. Depending on the agreement between the customer and the CMRA, the CMRA can forward the mail to the customer or hold it for pickup.

A customer may wish to use the services of a CMRA for privacy. For example, a person running a home-based business may not wish to divulge the home address. Alternatively, a customer in one community may contract with a CMRA in another community with a better known or more prestigious address.

Delivery Point Validation (DPV)

Delivery Point Validation (DPV®) is a United States Postal Service® (USPS®) technology that validates the accuracy of address information down to the individual mailing address. By using DPV® to validate addresses, you can reduce undeliverable-as-addressed (UAA) mail, thereby reducing postage costs and other business costs associated with inaccurate address information.

Without DPV®, the address validation process only verifies that an individual address is within a range of valid addresses for the given street. For example, the USPS data indicates that the range of addresses on Maple Lane is 500 to 1000. You attempt to validate an address of 610 Maple Ln Without DPV®, this address would appear to be valid because it is in the range of 500 to 1000. However, in reality the address 610 Maple Ln does not exist: the house numbers in this section of the street are 608, 609, 613, and 616. With DPV® processing, you would be alerted to the fact that 610 Maple Ln does not exist and you could take action to correct the address.

DPV® also provides unique address attributes to help produce more targeted mailing lists. For example, DPV® can indicate if a location is vacant and can identify commercial mail receiving agencies (CMRAs) and private mail boxes.

Although DPV® can validate the accuracy of an existing address, you cannot use DPV® to create address lists. For example, you can validate that 123 Elm Street Apartment 6 exists, but you cannot ask if there

is an Apartment 7 at the same street address. To prevent the generation of address lists, the DPV® database contains false positive records. False positive records are artificially manufactured addresses that reside in a false positive table. For each negative response that occurs in a DPV® query, a query is made to the false positive table. A match to this table will stop DPV® processing.

Early Warning System (EWS)

The Early Warning System (EWS) provides up-to-date address information for new and recently changed addresses that have not yet been updated in the monthly USPS database. EWS prevents address records from miscoding due to a delay in postal data reaching the USPS® databases.

The older the U.S. Postal Database, the higher potential you have for miscoding addresses. When a valid address is miscoded because the address it matches to in the U.S. Postal Database is inexact, it will result in a broken address.

EWS data consists of partial address information limited to the ZIP Code™, street name, predirectional, postdirectional, and a suffix. For an address record to be EWS-eligible, it must be an address not present on the most recent monthly production U.S. Postal Database.

The USPS® refreshes the EWS file on a weekly basis. You can download the EWS file from the USPS® website at **ribbs.usps.gov/files/CASS**.

Enhanced Line of Travel (eLot)

eLOT ensures that Enhanced Carrier Route mailings are sorted as close as possible to the actual delivery sequence. eLOT sequence is not an exact walk sequence but a sequence of ZIP + 4® Codes arranged in the order that the route is served by a carrier. First the ZIP + 4® groups are sequenced. Then the addresses within each group are identified as being in ascending or descending order.

eLOT can be used by mailers who prepare carrier route mailings other than high-density/125-piece or saturation mailings. eLOT sequencing is required for Basic Enhanced Carrier Route Standard Mail except automation-compatible, letter-size pieces.

Note: You must install the U.S. Postal database and eLOT database from the same month (i.e., September eLOT data must be processed with a September U.S. Postal database). If the U.S. Postal database and the eLOT database are not from the same month, there may be ZIP + 4® Codes for which eLOT numbers cannot be assigned. The ZIP Code™, ZIP + 4® Code, carrier route code, and the delivery point of an address must be provided to assign a eLOT code.

Locatable Address Conversion System (LACS)

The USPS® Locatable Address Conversion System (LACS) corrects addresses that have changed as a result of a rural route address converting to street-style address, a PO Box renumbering, or a street-style address changing. The following are examples of LACS^{Link} conversions:

- Rural Route Converted to Street-Style Address: Old Address: RR 3 Box 45 New Address: 1292 North Ridgeland Drive
- Street Renamed and Renumbered: Old Address: 23 Main Street New Address: 45 West First Avenue
- PO Box Renumbered: Old Address: PO Box 453 New Address: PO Box 10435

LACS^{Link} is required for CASS processing.

Residential Delivery Indicator (RDI)

Residential Delivery Indicator (RDI™) processing can help you determine the best cost for shipping your packages. RDI™ processing identifies whether an address is a business or a residential address. This

difference is important because some delivery services charge a higher price for delivery to a residential address than they do to a business address.

Street Name Aliases

Street name aliases in the United States are alternative names given to sections of a street. There are four types of street name aliases:

- **Preferred**—A preferred alias is the street name preferred locally. It typically applies only to a specific range of addresses on the street.
- **Abbreviated**—An abbreviated alias is a variation of the street name that can be used in cases where the length of AddressLine1 is longer than 31 characters. For example, the street name 1234 BERKSHIRE VALLEY RD APT 312A could be abbreviated to 1234 BERKSHIRE VLLY RD APT 312A.
- **Changed**—There has been an official street name change and the alias reflects the new name. For example if SHINGLE BROOK RD is changed to CANNING DR, then CANNING DR would be a changed alias type.
- **Other**—The street alias is made up of other names for the street or common abbreviations of the street.

The non-alias version of the street name is called the base street name.

Suite^{Link}

Suite^{Link} corrects secondary address information for U.S. business addresses whose secondary address information could not be validated. If Suite^{Link} processing is enabled, the firm name is matched to a database of known firm names and their secondary address information.

For example,

Firm Name: Pitney Bowes Software
Address Line 1: 4200 Parliament Place
Address Line 2: STE 1
Postal Code: 20706

In this case, Suite^{Link} processing would change the suite number to the correct suite number:

Firm Name: Pitney Bowes Software
Address Line 1: 4200 Parliament Place
Address Line 2: **STE 600**
Postal Code: 20706-1844

SuiteLink attempts to correct firm names in addresses where:

- A firm name is present
- A valid ZIP Code™, ZIP + 4® Code, and primary number could be determined
- A match has been made to a high-rise default record
- The secondary address information could not be validated through normal processing

Canadian Addressing Guidelines

Canada Post Corporation (CPC) address standards facilitate mail processing and timely delivery. The CPC addressing conventions include guidelines for:

- **Address Format** on page 731

- **Non-Address Data** on page 731
- **Addressee Information** on page 731
- **Delivery Address Information: Civic Address** on page 732
- **Delivery Address Information: Non-Civic Address** on page 732
- **Mode of Delivery Information** on page 732
- **Delivery Installation Information** on page 733
- **Municipality** on page 733
- **Province** on page 733
- **Postal Code** on page 733
- **Country** on page 734

Address Format

Canada Post defines a mailing address as the information required to identify a point of call and/or delivery point. The CPC sequence of components of an address block for mail originating in and addressed to a destination within Canada is:

1. Non-address data and addressee information
2. Delivery address information
3. Municipality, province, postal code.

Canadian addresses should follow these guidelines:

- Use the symbol for an address element rather than the full name.
- Use upper-case letters on all lines of the address block.
- Do not use the number sign (#) or the French equivalent (nº) in an address.
- Do not use punctuation, such as a comma, as a delimiter between address elements or components unless otherwise noted.
- Format all lines of the address with a flush-left margin.
- The bottom three lines of the address block must be: Addressee information, delivery address information, municipality, province, and postal code.

Non-Address Data

If there is any data that is extraneous, such as "Attention" or "Address Correction Required," this type of non-address data must always appear above the top line of the address block, above the delivery address information line.

Non-address data includes any additional information a mailer wishes to put on the mail piece. This includes words such as "Attention" and "Confidential" or account numbers and customer identification numbers.

Addressee Information

Addressee information identifies the intended person(s), firm, or establishment to which the piece of mail is addressed. Addressee information, with multiple lines of information, may include the following elements in the listed order:

- Name or in-care-of information
- Title
- Department or division name

- Company or establishment name.

The above order also applies for Large Volume Receivers (LVRs) and government departments or agencies. For departments with branches, branch names appear above the department name.

Delivery Address Information: Civic Address

Civic addressing generally applies to customers who receive door-to-door delivery or who receive their mail at a community mail box or "superbox." Unless otherwise noted, address components and elements on the same line should be separated from each other by one space.

Civic addresses may include the following elements:

- **Civic Number**—The official number that has been assigned to that address by the municipality.
- **Civic Number Suffix**—An alpha character or fraction appearing after the civic number that must be included in the address, such as 11D JESSOME ST, or 91 1/2 KING ST.
- **Street Name**—The official street name as it is recognized by the municipality, with no translation. If the street name is a number followed by an ordinal such as "ST", "ND", "TH", or "E", as in "1ST", "2ND", "3RD", "4TH" or "1E", "2E," there will be no space between the number and the ordinal. If the street name is alphanumeric, there will be no space between the numeric and the alpha portion (14B ST instead of 14 B ST).
- **Street Type**—When an address is presented in English, the street type always follows the street name. When an address is presented in French, the street type appears before the street name (RUE RENE-LEVESQUE) unless the street name is an ordinal. In this case, the street type follows the street name (2061 36E RUE). The street type is always identified by CPC symbols.
- **Street Direction**—Where required, it should be identified by a one- or two-character symbol (such as N, S, NE, NW, O, NO) and located as the last element of street information.
- **Unit Designator**—The unit designator identifies the specific unit as assigned by the building. The value can be alpha, alphanumeric, or numeric. Where the unit identifier is a number, it is presented in numeric format ("2" instead of the word two). If the unit designator is not used in the address, the unit designator is placed before the street information separated by a hyphen (317-10228 148 ST). Valid examples for placement of the unit designator include: 1 MAIN ST APT 1 or APT 1 1 MAIN ST or 1-1 MAIN ST.

Delivery Address Information: Non-Civic Address

Non-civic addresses are used when the service provided to a customer is a lock box, general delivery, or route service where civic address information is not available.

Mode of Delivery Information

Mode of delivery information is to be placed on the same line as delivery installation information. Mode of delivery information may consist of the following elements:

- **Mode of Delivery Designator**—The official CPC symbol for Postal Box (PO BOX), Rural Route (RR), Suburban Service (SS), Mobile Route (MR), or General Delivery (GD).
- **Mode of Delivery Identifier**—Numeric identifier separated from the mode of delivery designator by one space. Do not use the number sign (#) or the French abbreviation (nº) before the mode of delivery identifier (PO BOX 123, not PO BOX #123).

Delivery Installation Information

In areas where there is more than one delivery facility in the municipality, use delivery installation information with mode of delivery to identify the specific facility servicing the delivery address. Place delivery information after the mode of delivery, separated by one space.

Delivery installation information may consist of these elements:

- **Delivery Installation Area Name**—The area name identifies the general location of the delivery installation and is usually a village, town, community, municipality, or metropolitan area. The delivery installation area name is not required if it is the same as the municipality name. For example:

  Correct:

  PO BOX 59
  BELL ISLAND FRONT NF A0A 1H0

  Incorrect:

  PO BOX 59 BELL ISLAND FRONT
  BELL ISLAND FRONT NF A0A 1H0

- **Delivery Installation Type and Qualifier**—The type and qualifier identify the specific delivery installation in urban areas where there is more than one installation. For example, in the delivery address GD SCARBOROUGH STN LCD WEST HILL, the delivery installation area name is SCARBOROUGH, the delivery installation type is STN, and the delivery installation qualifier is LCD WEST HILL.

Municipality

Municipality refers to a municipality, town, village, community, or directory area name recognized as a valid mailing destination.

The municipality is the first component of the last line of the Canadian address and is required for all address types.

Province

The province component is required for all address types. It is the second component of the last line of a Canadian address. It should appear on the same line as the municipality, separated from it by a single space.

Postal Code

The postal code component is required for all address types and is made up of six alphanumeric characters arranged in two sets of three. The first set of three (letter, number, letter) is known as the Forward Sortation Area (FSA). The second set of three characters (number, letter, number) is known as the Local Delivery Unit (LDU).

The format of the postal code is always A9A 9A9, where A is a letter and 9 is a number. There should always be one space between the first three and the last three characters of the postal code.

The postal code should appear on the same line as the municipality and the province and is separated from the province by at least two spaces.

Country

The country name is used only on mail pieces delivered outside Canada. The official English or French language spelling for the country name should appear alone on the last line of the address block for a foreign address.

Mail originating in other countries for delivery in Canada should have CANADA appear alone on the last line of the address block.

Australian Addressing

In Australia, Australia Post sets addressing standards. For a list of Australia Post publications that describe these standards and the Address Matching Approval System (AMAS) program, visit the Australia Post web site at **www.auspost.com.au**.

International Addressing

Addressing products from Pitney Bowes Software classify addresses outside the U.S. and Canada as "international" addresses. The standards for international addresses vary from country to country. Pitney Bowes Software has compiled these standards into an international address coding system that can validate and correct address elements for approximately 86 countries and validate postal code formats for 141 countries.

Information about the postal standards of Universal Postal Union member countries can be found online at the Universal Postal Union website **www.upu.int**.

AutoCompleteLoqate

AutoCompleteLoqate offers real-time entry of address data for fast, accurate results. Users are returned instant results based on each character entered into the form, ensuring only accurate data is entered into the database.

Input

The following table lists the input for AutoCompleteLoqate.

Table 223: Input Format

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
City	The city name.

columnName	Description / Valid Values
Country	The country code or name, in any of the following formats: • 2-digit ISO country code • 3-digit UPU Country code • English country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	The company or firm name.
PostalCode	The postal code for the address.
StateProvince	The state or province.

Options

Table 224: AutoCompleteLoqate Options

optionName	Description/Valid Values
Database	Specifies the database to be used for address processing. Only databases that have been defined in the **Database Resources** panel in the Management Console are available.
MaximumResults	The maximum number of addresses that AutoCompleteLoqate should return. The default is 10.
Match level	Specifies the level of constraint to apply when matching addresses. One of the following: **Distinct** Returns every distinct set of requested output fields found in the query. If two or more output field sets have the same value in every field, only one set from that group of duplicate sets is returned; the output records obtained are not repeated. This setting can return multiple records. **Unconstrained** Returns all output address record(s) without any modification. **Unique** Returns only those records with values that are unique among the results of the query. This setting can return

optionName	Description/Valid Values
	either one record or no records. If no result was returned from the query then no output record is returned. **Note:** Entering information into one field, use the Distinct setting to return multiple results into the drop-down list. After selecting one result from the drop-down list, conduct a secondary search using the Unique setting to auto-fill other fields.
Alias output	Returns output record(s) matching an alias for fields that do not return a range of values. These might include Administrative Area, SubAdministrative Area, Locality, and so on. For example, a valid alias for the locality “SOUTH SAN FRANCISCO” might be "S SAN FRAN". One of the following: **Default** Returns records corresponding to all aliases for that field. **Exhaustive** Returns a set of output record(s) for all combinations of every alias. Separate records are returned for each alias. **First** From the list of output variables selected, returns the first alias found in the list of aliases corresponding to that field.
Range output	Returns output record(s) matching an alias for fields that return a range of values, such as Premise Number. One of the following: **Default** Recommended in cases that display the range value that contains the queried value. **Full** Recommended in cases that display each distinct value explicitly instead of displaying a range of values. This setting expands the range into its individual results that comprise that range. **Match** Recommended in cases that display individual values extracted from a range of results. This setting allows you to search for values included within a range of values, such as a specific PremiseNumber.

optionName	Description/Valid Values
	Range Recommended in cases where the verified output record(s) must be displayed in a human-readable range format (e.g., start and end range values separated by a dash). Results may also include an “odd” or “even” text, meaning the values in that range are odd or even values only.
Default country	Specifies the default country. You should specify the country where most of your addresses reside. For example, if most of the addresses you process are in Canada, specify Canada. The valid country names are: Afghanistan, Albania, Algeria, American Somoa, Andorra, Angola, Anguilla, Antigua And Barbuda, Argentina, Armenia, Aruba, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bermuda, Bhutan, Bolivia, Bosnia And Herzegovina, Botswana, Brazil, British Virgin Islands, Brunei Darussalam, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon, Canada, Cape Verde, Cayman Islands, Central African Republic, Chad, Chile, China, Colombia, Comoros Islands, Congo, Cook Islands, Costa Rica, Cote D'Ivoire, Croatia, Cuba, Cyprus, Czech Republic, Democratic Republic Of Congo, Denmark, Djibouti, Dominica, Dominican Republic, East Timor, Ecuador, Egypt, El Salvador, Equitorial Guinea, Eritrea, Estonia, Ethiopia, Falkland Islands, Faroe Islands, Federated States Of Micronesia, Fiji, Finland, France, French Guiana, Gabon, Gambia, Germany, Ghana, Gibraltar, Greece, Greenland, Grenada, Guadeloupe, Guam, Guatemala, Guinea, Guinea Bissau, Guyana, Haiti, Holy See, Honduras, Hong Kong, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Korea, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania, Luxembourg, Macau, Macedonia, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall Islands, Martinique, Mauritania, Mauritius, Mayotte, Mexico, Moldova, Monaco, Mongolia, Monserrat, Morocco, Mozambique, Myanmar, Namibia, Nauru, Nepal, Netherlands Antilles, New Caledonia, New Zealand, Nicaragua, Niger, Nigeria, Niue, Norway, Oman, Pakistan, Palau, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Pitcairn Islands, Poland, Portugal, Puerto Rico, Qatar, Republic Of Georgia, Republic Of Korea, Republic Of Singapore, Reunion, Romania, Russia, Rwanda, Saint Helena, Saint Kitts And Nevis, Saint Lucia, Saint Pierre And Miquelon, Saint Vincent And The Grenadines, Samoa, San Marino, Sao Tome And Principe, Saudi Arabia, Senegal, Seychelles, Sierra Leone, Slovakia, Slovenia, Solomon Islands, Somalia, South Africa, Spain, Sri Lanka, Sudan, Surivalue, Swaziland, Sweden, Switzerland, Syria, Tahiti, Taiwan, Tajikistan, Tanzania, Thailand, The Netherlands, Togo, Tonga, Trinidad

optionName	Description/Valid Values
	And Tobago, Tristan Da Cunha, Tunisia, Turkey, Turkmenistan, Turks And Caicos Islands, Tuvalu, Uganda, Ukraine, United Arab Emirates, United Kingdom, United States, Uruguay, Uzbekistan, Vanuatu, Venezuela, Vietnam, Virgin Islands (US), Wallis And Futuna, Yemen, Yugoslavia, Zambia, Zimbabwe
Data license error handling	Specifies how you want Spectrum Technology Platform to respond when a data license error occurs. **Fail the job** — Fail the entire job if a data license error occurs. **Fail the record** — Fail the record(s) for which the data license error occurs and continue processing.
Output Fields	Specifies which fields should be included in the output. Check the box for each field you want to return in the output. Selecting **All fields** will result in every field listed being returned in the output.

Output

The output from AutoCompleteLoqate is optional and corresponds directly to the fields you selected in the Output Fields section of the AutoCompleteLoqate Options dialog box.

Table 225: AutoCompleteLoqate Output

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
City	The city name.
Country	The three-character ISO 3116-1 Alpha-3 code for the country. For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	The firm name.
HouseNumber	The ending house number for the range in which the candidate address's house number falls.

columnName	Description / Valid Values
PostalCode	The postal code.
PostalCode.AddOn	The last four digits of the ZIP + 4® Code.
ProcessedBy	Indicates which address coder processed the address. **LOQATE** The Loqate coder processed the address.
StateProvince	The state or province abbreviation.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	The reason for failure, if there is one. • DisabledCoder • RequestFailed • NoLookupAddressFound
Status.Description	A description of the problem, if there is one. **Did not return multiples** The input address matched only one address in the database. AutoCompleteLoqate returns data only if multiple possible matches were found. **Not able to look up the address pattern** AutoCompleteLoqate is not able to process the partial address.

AutoCompleteLoqate Web Service

In addition to using Auto Complete Loqate in Enterprise Designer and Management Console, you can also access a sample web application that demonstrates the Auto Complete Loqate functionality. When you enter a partial address, this application makes a call to the Auto Complete Loqate REST web service, which returns a suggested address.

Note: Prior to using this feature, you must add an Auto Complete Loqate database resource in Management Console and save the database resource in the Auto Complete Loqate Service.

1. Be sure the Spectrum™ Technology Platform server is running.
2. Open a web browser and go to:

http://<servername>:<port>/autocomplete. For example, if your server is named "myserver" and it uses the default HTTP port 8080, you would go to:

http://myserver:8080/autocomplete.

Note: This site is best viewed in Internet Explorer 8.0 or later, Chrome, or Mozilla Firefox.

3. When the login screen appears, enter "**guest**" as the user name and leave the password field blank.
4. Press **OK**.
5. Select a country from the drop-down list.
6. Begin typing your address in any of the fields provided.
7. Select from the list of suggested addresses.
8. To begin a new call, click **Search Again**, which will clear the fields you used in your previous call.

GetCandidateAddresses

GetCandidateAddresses returns a list of addresses that are considered matches for a given input address. GetCandidateAddresses returns candidate addresses only if the input address matches multiple addresses in the postal database. If the input address matches only one address in the postal database, then no address data is returned.

For addresses outside the U.S. and Canada, you may notice inconsistent results between the multiple matches returned by ValidateAddress and the results for that same address returned by GetCandidateAddresses. If you experience inconsistent results, it is likely because you set the performance tuning setting in ValidateAddress to a value other than 100. To obtain consistent results between GetCandidateAddresses and ValidateAddress, set the performance tuning option to 100.

Note: By default, GetCandidateAddresses does not match to individual house numbers. Rather, it uses house number ranges for each street. After GetCandidateAddresses has determined the street name, city name, state/province name, and postal code, it checks to make sure the input house number falls within one of the ranges of house numbers given for the matched street name. The same type of logic applies to unit numbers. If you want to determine that an individual house number is valid, you should use the ValidateAddress Delivery Point Validation (DPV) processing option. DPV processing is only available for U.S. addresses.

The Canadian coder contains a reverse lookup routine that takes as input a specific postal code and returns the street information stored in the database for that postal code. To use this function enter nothing but a Canadian postal code in the PostalCode field. See the second example to view the return from a sample postal code.

GetCandidateAddresses is part of the Universal Addressing Module. For more information on this module, including other components included with it, see **What Is the Universal Addressing Module?** on page 722.

U.S. Address Example

AddressLine1: PO Box 1 City: NY State: NY

Preview Output

AddressLine1	PO Box 1	PO Box 1
City	New York	New York
Country	USA	USA
HouseNumberHigh	60	9
HouseNumberLow	1	1
HouseNumberParity	B	B
MatchLevel	A	A
PostalCode	10002	10008
PostalCode.AddOn	0001	0001
ProcessedBy	USA	USA
RecordType	PostOfficeBox	PostOfficeBox
RecordType.Default		
StateProvince	NY	NY
UnitNumberParity		

Canadian Address Example

PostalCode: A1A1A1

Preview Output

AddressLine1	LOWER BATTERY RD	LOWER BATTERY RD
City	ST. JOHN'S	ST. JOHN'S
Country	CAN	CAN
HouseNumberHigh	000003	000004 A
HouseNumberLow	000001	000002
HouseNumberParity	O	E
MatchLevel	A	A
PostalCode	A1A1A1	A1A1A1
ProcessedBy	CAN	CAN
RecordType	Normal	Normal
StateProvince	NL	NL

Input

The following table lists the input for GetCandidateAddresses.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 226: Input Format

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
AddressLine3	The third address line. Does not apply to U.S. and Canadian addresses.
AddressLine4	The fourth address line. Does not apply to U.S. and Canadian addresses.
AddressLine5	The fifth address line. Applies only to U.K. addresses. May contain street name, unit number, building number, and so on.
City	The city name.
StateProvince	The state or province. For U.S. addresses only, you may put the state in the City field instead of the StateProvince field.
PostalCode	The postal code for the address. For U.S. addresses this is the ZIP Code™ in one of the following formats: 99999 99999-9999 A9A9A9 A9A 9A9 9999 999 **Note:** For Canadian addresses you can complete just this field and have candidate address data returned. For other countries, AddressLine1 and AddressLine2 must also be completed.
Country	The country code or name, in any of the following formats: • 2-digit ISO country code • 3-digit UPU Country code • English country name • French country name • German country name

columnName	Description / Valid Values
	• Spanish country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	The company or firm name.
USUrbanName	U.S. address urbanization name. Used primarily for Puerto Rico addresses.

Options

Table 227: GetCandidateAddresses Options

optionName	Description/Valid Values
PerformUSProcessing	Specifies whether or not to process U.S. addresses. If you enable U.S. address processing GetCandidateAddresses will attempt to retrieve candidate addresses for U.S. addresses. If you disable U.S. address processing, U.S. addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for U.S. address processing you must disable U.S. address processing in order for your jobs to complete successfully, regardless of whether or not they contain U.S. addresses. **Note:** You must have a valid license for U.S. address processing to successfully process U.S. addresses. If you enable U.S. address processing but are not licensed for this feature, or your license has expired, you will receive an error. **Y** Yes, process U.S. addresses (default). **N** No, do not process U.S. addresses.
Database.US	Specifies the database to be used for U.S. address processing. Only databases that have been defined in the **US Database Resources** panel in the Management Console are available.

optionName	Description/Valid Values
PerformCanadianProcessing	Specifies whether or not to process Canadian addresses. If you enable Canadian address processing GetCandidateAddresses will attempt to retrieve candidate addresses for Canadian addresses. If you disable Canadian address processing, Canadian addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for Canadian address processing you must disable Canadian address processing in order for your jobs to complete successfully, regardless of whether or not they contain Canadian addresses. **Note:** You must have a valid license for Canadian address processing to successfully process Canadian addresses. If you enable Canadian address processing but are not licensed for this feature, or your license has expired, you will receive an error. **Y** Yes, process Canadian addresses (default). **N** No, do not process Canadian addresses.
Database.Canada	Specifies the database to be used for Canadian address processing. Only databases that have been defined in the **Canadian Database Resources** panel in the Management Console are available.
PerformInternationalProcessing	Specifies whether or not to process international addresses (addresses outside the U.S. and Canada). If you enable international address processing GetCandidateAddresses will attempt to retrieve candidate addresses for international addresses. If you disable international address processing, international addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for international address processing you must disable international address processing in order for your jobs to complete successfully, regardless of whether or not they contain international addresses. **Note:** You must have a valid license for international address processing to successfully process international addresses. If you enable international address processing but are not licensed for this feature, or your license has expired, you will receive an error.

optionName	Description/Valid Values
	Y Yes, process international addresses (default). **N** No, do not process international addresses.
Database.International	Specifies the database to be used for international address processing. Only databases that have been defined in the **International Database Resources** panel in the Management Console are available.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output is in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output is in upper case. For example: 123 MAIN ST MYTOWN FL 12345
MaximumResults	The maximum number of candidate addresses that GetCandidateAddresses should return. The default is 10. The maximum is 10.
OutputShortCityName	For U.S. addresses, specifies whether or not to return the USPS®-approved abbreviation for the city, if there is one. The USPS® provides abbreviations for city names that are 14 characters long or longer. City abbreviations are 13 characters or less and can be used when there is limited space on the mailing label. If there is no short city name for the city, then the full city name is returned. The valid values are: **Y** Yes, return the short city name. **N** No, do not return the short city name.
DualAddressLogic	(U.S. addresses only). Controls whether GetCandidateAddresses should return a street match or a PO Box/Rural Route/Highway Contract match when the address contains both street and PO Box/Rural Route/Highway Contract information. For more information, see **About Dual Address Logic** on page 781.

optionName	Description/Valid Values
	N (Default) USPS® CASS™ regulations determine the address returned based on the following order of priority: **1.** PO Box **2.** Firm **3.** Highrise **4.** Street **5.** Rural Route **6.** General Delivery **S** Return a street match, regardless of the address line. **P** Return a PO Box match, regardless of the address line.
StreetMatchingStrictness	The strictness of the street name match (U.S. addresses only). **E** The input street name must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium" (default). **L** The matching algorithm is "loose".
FirmMatchingStrictness	The strictness of the firm name match (U.S. addresses only). **E** The input firm name must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium" (default). **L** The matching algorithm is "loose."
DirectionalMatchingStrictness	The strictness of the directional match. **E** The input directional must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium" (default). **L** The matching algorithm is "loose."

optionName	Description/Valid Values
PerformESM	Specifies whether or not to perform Enhanced Street Matching (ESM). ESM applies extra matching logic with additional data to any input address that is not matched through the regular address validation process. ESM applies to U.S. addresses only. The valid values are: **Y** Yes, perform ESM processing. **N** No, do not perform ESM processing (default).
AddressLineSearchOnFail	Specifies whether ValidateAddress will search address lines for the city, state/province, and postal code. This option enables ValidateAddress to search the AddressLine input fields for the city, state/province, postal code, and country when the address cannot be matched using the values in the City, StateProvince, and PostalCode input fields. Consider enabling this option if your input addresses have the city, state/province, and postal code information in the AddressLine fields. Consider disabling this option if your input addresses use the City, State/Province and PostalCode fields. If you enable this option and these fields are used, there is an increased possibility that ValidateAddress will fail to correct values in these fields (for example a misspelled city name). One of the following: **Y** Yes, search the address line fields (default). **N** No, do not search the AddressLine fields.

Output

GetCandidateAddresses returns the following output.

Note: The output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 228: GetCandidateAddresses Output

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
AddressLine3	The third address line.
AddressLine4	The fourth address line.
AddressLine5	For U.K. addresses only. If the address was validated, the fifth line of the validated and standardized address. If the address could not be validated, the fifth line of the input address without any changes.
City	The city name.
Country	The three-character ISO 3116-1 Alpha-3 code for the country. For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	The firm name.
HouseNumberHigh	The ending house number for the range in which the candidate address's house number falls.
HouseNumberLow	The beginning house number for the range in which the candidate address's house number falls.
HouseNumberParity	Indicates the numbering scheme for the house numbers between HouseNumberLow and HouseNumberHigh, as follows: **E** Only even values **O** Only odd values **B** Both
MatchLevel	For addresses outside the U.S. and Canada, identifies the match level for the candidate address. U.S. and Canadian addresses are always "A." One of the following: **A** The candidate matches the input address at the street level.

columnName	Description / Valid Values
	B The candidate matches the input address at the state/province level.
PostalCode	The postal code. In the U.S. this is the ZIP Code™.
PostalCode.AddOn	The last four digits of the ZIP + 4® Code. U.S. addresses only.
RecordType	The type of address record, as defined by U.S. and Canadian postal authorities (U.S. and Canadian addresses only): • FirmRecord • GeneralDelivery • HighRise • PostOfficeBox • RRHighwayContract • Normal
RecordType.Default	Code indicating the "default" match: **Y** The address matches a default record. **null** The address does not match a default record.
StateProvince	The state or province abbreviation.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	The reason for failure, if there is one. There is only one possible value: • DisabledCoder • RequestFailed
Status.Description	A description of the problem, if there is one. **Did not return multiples** The input address matched only one address in the database. GetCandidateAddresses only returns data if multiple possible matches were found.

columnName	Description / Valid Values	
	Number of candidates is not greater than 1	The input address matched more than one address in the database but no addresses were returned.
	PerformUSProcessing disabled	This value will appear if Status.Code=DisabledCoder.
	PerformCanadianProcessing disabled	This value will appear if Status.Code=DisabledCoder.
	PerformInternationalProcessing disabled	This value will appear if Status.Code=DisabledCoder.
UnitNumberHigh	The ending unit number for the range in which the candidate address's unit number falls.	
UnitNumberLow	The beginning unit number for the range in which the candidate address's unit number falls.	
UnitNumberParity	Indicates the numbering scheme for the unit numbers between UnitNumberLow and UnitNumberHigh, as follows:	
	E	Only even values
	O	Only odd values
	B	Both
USUrbanName	The validated city urbanization name. Urbanization names are used primarily for Puerto Rico addresses.	

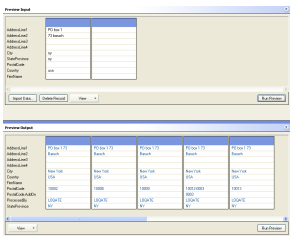
GetCandidateAddressesLoqate

GetCandidateAddressesLoqate returns a list of addresses that are considered matches for a given input address. GetCandidateAddressesLoqate returns candidate addresses only if the input address matches multiple addresses in the postal database. If the input address matches only one address in the postal database, then no address data is returned. The Country input field is required; if this field is blank, no output will be returned.

Note: By default, GetCandidateAddressesLoqate does not match to individual house numbers. Rather, it uses house number ranges for each street. After GetCandidateAddressesLoqate has determined the street name, city name, state/province name, and postal code, it checks to make sure the input house number falls within one of the ranges of house numbers given for the matched street name. The same type of logic applies to unit numbers.

GetCandidateAddressesLoqate is part of the Universal Addressing Module. For more information on this module, including other components included with it, see **What Is the Universal Addressing Module?** on page 722

U.S. Address Example



Canadian Address Example

PostalCode: A1A1A1

Preview Output

AddressLine1	LOWER BATTERY RD	LOWER BATTERY RD
City	ST. JOHN'S	ST. JOHN'S
Country	CAN	CAN
HouseNumberHigh	000003	000004 A
HouseNumberLow	000001	000002
HouseNumberParity	O	E
MatchLevel	A	A
PostalCode	A1A1A1	A1A1A1
ProcessedBy	CAN	CAN
RecordType	Normal	Normal
StateProvince	NL	NL

Input

The following table lists the input for GetCandidateAddressesLoqate.

Table 229: Input Format

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
AddressLine3	The third address line.

columnName	Description / Valid Values
AddressLine4	The fourth address line.
City	The city name.
Country	The country code or name, in any of the following formats: • 2-digit ISO country code • 3-digit UPU Country code • English country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908. **Note:** This field is required. If this field is blank, no output will be returned.
FirmName	The company or firm name.
PostalCode	The postal code for the address. For U.S. addresses this is the ZIP Code™ in one of the following formats:
StateProvince	The state or province. For U.S. addresses only, you may put the state in the City field instead of the StateProvince field.

Options

Table 230: GetCandidateAddressesLoqate Options

optionName	Description/Valid Values
Database.US	Specifies the database to be used for address processing. Only databases that have been defined in the Management Console are available.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output is in mixed case (default). For example: 123 Main St Mytown FL 12345

optionName	Description/Valid Values
	U The output is in upper case. For example: 123 MAIN ST MYTOWN FL 12345
DefaultCountry	Specifies the default country. You should specify the country where most of your addresses reside. For example, if most of the addresses you process are in Canada, specify Canada. GetCandidateAddressLoqate uses the country you specify to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields. The valid country names are: Afghanistan, Albania, Algeria, American Somoa, Andorra, Angola, Anguilla, Antigua And Barbuda, Argentina, Armenia, Aruba, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bermuda, Bhutan, Bolivia, Bosnia And Herzegovina, Botswana, Brazil, British Virgin Islands, Brunei Darussalam, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon, Canada, Cape Verde, Cayman Islands, Central African Republic, Chad, Chile, China, Colombia, Comoros Islands, Congo, Cook Islands, Costa Rica, Cote D'Ivoire, Croatia, Cuba, Cyprus, Czech Republic, Democratic Republic Of Congo, Denmark, Djibouti, Dominica, Dominican Republic, East Timor, Ecuador, Egypt, El Salvador, Equitorial Guinea, Eritrea, Estonia, Ethiopia, Falkland Islands, Faroe Islands, Federated States Of Micronesia, Fiji, Finland, France, French Guiana, Gabon, Gambia, Germany, Ghana, Gibraltar, Greece, Greenland, Grenada, Guadeloupe, Guam, Guatemala, Guinea, Guinea Bissau, Guyana, Haiti, Holy See, Honduras, Hong Kong, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Korea, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania, Luxembourg, Macau, Macedonia, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall Islands, Martinique, Mauritania, Mauritius, Mayotte, Mexico, Moldova, Monaco, Mongolia, Monserrat, Morocco, Mozambique, Myanmar, Namibia, Nauru, Nepal, Netherlands Antilles, New Caledonia, New Zealand, Nicaragua, Niger, Nigeria, Niue, Norway, Oman, Pakistan, Palau, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Pitcairn Islands, Poland, Portugal, Puerto Rico, Qatar, Republic Of Georgia, Republic Of Korea, Republic Of Singapore, Reunion, Romania, Russia, Rwanda, Saint Helena, Saint Kitts And Nevis, Saint Lucia, Saint Pierre And Miquelon, Saint Vincent And The Grenadines, Samoa, San Marino, Sao Tome And Principe, Saudi Arabia, Senegal, Seychelles, Sierra Leone, Slovakia, Slovenia, Solomon Islands, Somalia, South Africa, Spain, Sri Lanka, Sudan, Surivalue, Swaziland, Sweden, Switzerland, Syria, Tahiti, Taiwan, Tajikistan, Tanzania, Thailand, The Netherlands, Togo, Tonga, Trinidad

optionName	Description/Valid Values	
	And Tobago, Tristan Da Cunha, Tunisia, Turkey, Turkmenistan, Turks And Caicos Islands, Tuvalu, Uganda, Ukraine, United Arab Emirates, United Kingdom, United States, Uruguay, Uzbekistan, Vanuatu, Venezuela, Vietnam, Virgin Islands (US), Wallis And Futuna, Yemen, Yugoslavia, Zambia, Zimbabwe	
OutputScript/Alphabet	Specifies the alphabet or script in which the output should be returned. This option is bi-directional and generally takes place from Native to Latin and Latin to Native.	
	Input	Do not perform transliteration and provide output in the same script as the input (default).
	Native	Output in the native script for the selected country wherever possible.
	Latn	Use English values.
	Cyrl	Use Cyrillic values.
	Grek	Use Greek values.
	Hebr	Use Hebrew values.
	Hani	Use Kanji values.
	Hans	Use Simplified Chinese values.
	Arab	Use Arabic values.
	Thai	Use Thai values.
	Hang	Use Hangul values.
MaximumResults	The maximum number of candidate addresses that GetCandidateAddressesLoqate should return. The default is 10. The maximum is 99.	
Data license error handling	Specifies how you want Spectrum Technology Platform to respond when a data license error occurs.	
	Fail the job	Fail the entire job if a data license error occurs.
	Fail the record	Fail the record(s) for which the data license error occurs and continue processing.

Output

GetCandidateAddressesLoqate returns the following output.

Table 231: GetCandidateAddressesLoqate Output

columnName	Description / Valid Values
AddressLine1	The first address line.
AddressLine2	The second address line.
AddressLine3	The third address line.
AddressLine4	The fourth address line.
City	The city name.
Country	The three-character ISO 3116-1 Alpha-3 code for the country. For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	The firm name.
PostalCode	The postal code. In the U.S. this is the ZIP Code™.
PostalCode.AddOn	The last four digits of the ZIP + 4® Code. U.S. addresses only.
ProcessedBy	Indicates which address coder processed the address. **LOQATE** The Loqate coder processed the address.
StateProvince	The state or province abbreviation.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	The reason for failure, if there is one. There is only one possible value: • RequestFailed
Status.Description	A description of the problem, if there is one. There is only one possible value: **Did not return multiples** The input address matched only one address in the database. GetCandidateAddressesLoqate

columnName	Description / Valid Values
	only returns data if multiple possible matches were found.

GetCityStateProvince

GetCityStateProvince returns a city and state/province for a given input postal code.

Note: GetCityStateProvince works with U.S. and Canadian addresses only.

GetCityStateProvince is part of the Universal Addressing Module. For more information on this module, including other components included with it, see **What Is the Universal Addressing Module?** on page 722.

Input

The following table shows the input fields.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section

Table 232: GetCityStateProvince Input

columnName	Description / Valid Values
PostalCode	A U.S. ZIP Code™ or Canadian postal code in one of the following formats: 99999 99999-9999 A9A9A9 A9A 9A9

Options

Table 233: GetCityStateProvince Options

optionName	Description / Valid Values
PerformUSProcessing	Specifies whether or not to process U.S. addresses. If you enable U.S. address processing GetCityStateProvince will attempt to return the state for U.S. addresses. If you disable U.S. address processing, U.S. addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for U.S. address processing you must disable U.S. address processing in order for your jobs to complete successfully, regardless of whether or not they contain U.S. addresses. **Note:** You must have a valid license for U.S. address processing to successfully process U.S. addresses. If you enable U.S. address processing but are not licensed for this feature, or your license has expired, you will receive an error. One of the following: **Y** Yes, process U.S. addresses (default). **N** No, do not process U.S. addresses.
Database.US	Specifies the database to be used for U.S. address processing. Only databases that have been defined in the **US Database Resources** panel in the Management Console are available.
PerformCanadianProcessing	Specifies whether or not to process Canadian addresses. If you enable Canadian address processing GetCityStateProvince will attempt to return the province for Canadian addresses. If you disable Canadian address processing, Canadian addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for Canadian address processing you must disable Canadian address processing in order for your jobs to complete successfully, regardless of whether or not they contain Canadian addresses. **Note:** You must have a valid license for Canadian address processing to successfully process Canadian addresses. If you enable Canadian address processing but are not licensed for this feature, or your license has expired, you will receive an error. **Y** Yes, process Canadian addresses (default).

optionName	Description / Valid Values
	N No, do not process Canadian addresses.
Database.Canada	Specifies the database to be used for Canadian address processing. Only databases that have been defined in the **Canadian Database Resources** panel in the Management Console are available.
OutputVanityCity	Specifies whether or not to include non-mailing city names in the output. A non-mailing city name is an alternate name for the primary city name. For example, Hollywood is a non-mailing city name for Los Angeles. **Y** Yes, include non-mailing city names. **N** No, do not include non-mailing city names (default).
MaximumResults	Specifies the maximum number of city-state/province pairs to return. The default value is 10.

Output

GetCityStateProvince returns the matching city and state/province for the input postal code as well as a code to indicate the success or failure of the match attempt. If more than one city/state or city/province matches the input postal code, multiple output records are returned.

Note: The output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 234: GetCityStateProvince Output

columnName	Description / Valid Values
City	The matched city name.
City.Type	The USPS® standardized city name type (U.S. addresses only). **V** Vanity (non-mailing) city name. **P** Primary. The city name is the primary mailing city name. **S** Secondary. The city name is an alternate city name but is acceptable. A city can have multiple secondary city names.
PostalCode	The input postal code.

columnName	Description / Valid Values
ProcessedBy	Indicates which address coder processed the address. One of the following: **USA** The U.S. address coder processed the address. **CAN** The Canadian address coder processed the address.
StateProvince	The state or province abbreviation.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	The reason for failure, if there is one. The only valid value is: • DisabledCoder • UnrecognizedPostalCode
Status.Description	The description of the failure. The valid values are: **Postal code not found** This value will appear if Status.Code=UnrecognizedPostalCode. **PerformUSProcessing disabled** This value will appear if Status.Code=DisabledCoder. **PerformCanadianProcessing disabled** This value will appear if Status.Code=DisabledCoder.

GetCityStateProvinceLoqate

GetCityStateProvinceLoqate returns a city and state/province for a given input postal code.

This stage is part of the Universal Addressing Module. For more information on this module, including other components included with it, see **What Is the Universal Addressing Module?** on page 722

Input

The following table shows the input fields.

Table 235: GetCityStateProvinceLoqate Input

columnName	Description / Valid Values
Country	The country code or name, in any of the following formats: • 2-digit ISO country code • 3-digit UPU Country code • English country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
PostalCode	The postal code for the address.

Options

Table 236: GetCityStateProvinceLoqate Options

optionName	Description / Valid Values
Database.US	Specifies the database to be used for processing. Only databases that have been defined in the Management Console are available.
MaximumResults	Specifies the maximum number of city-state/province pairs to return. The default value is 10.
OutputScript/Alphabet	Specifies the alphabet or script in which the output should be returned. This option is bi-directional and generally takes place from Native to Latin and Latin to Native. **Input** — Do not perform transliteration and provide output in the same script as the input (default). **Native** — Output in the native script for the selected country wherever possible. **Latn** — Use English values. **Cyrl** — Use Cyrillic values. **Grek** — Use Greek values. **Hebr** — Use Hebrew values. **Hani** — Use Kanji values. **Hans** — Use Simplified Chinese values.

optionName	Description / Valid Values	
	Arab	Use Arabic values.
	Thai	Use Thai values.
	Hang	Use Hangul values.
Data license error handling	Specifies how you want Spectrum Technology Platform to respond when a data license error occurs.	
	Fail the job	Fail the entire job if a data license error occurs.
	Fail the record	Fail the record(s) for which the data license error occurs and continue processing.

Output

GetCityStateProvinceLoqate returns the matching city and state/province for the input postal code as well as a code to indicate the success or failure of the match attempt. If more than one city/state or city/province matches the input postal code, multiple output records are returned.

Table 237: GetCityStateProvinceLoqate Output

columnName	Description / Valid Values
City	The matched city name.
Country	The country in the format determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English
PostalCode	The input postal code.
ProcessedBy	Indicates which address coder processed the address. **LOQATE** The Loqate coder processed the address.
StateProvince	The state or province abbreviation.
Status	Reports the success or failure of the match attempt.

columnName	Description / Valid Values
	null Success **F** Failure
Status.Code	The reason for failure, if there is one. The only valid value is: • UnrecognizedPostalCode
Status.Description	The description of the failure. The only valid value is: **Postal code not found** This value will appear if Status.Code=UnrecognizedPostalCode.

GetPostalCodes

GetPostalCodes allows you to look up the postal codes for a particular city. The service takes a city, state, and country as input and returns the postal codes for that city. The input must be exactly correct in order to return postal codes.

Note: GetPostalCodes only works with U.S. addresses.

GetPostalCodes is part of the Universal Addressing Module. For more information on this module, including a listing of other components included with it, see **What Is the Universal Addressing Module?** on page 722.

Input

GetPostalCodes takes a city, state/province, and country as input.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 238: GetPostalCodes Input

columnName	Description / Valid Values
City	The city whose postal codes you want to look up. You may put the city and state in the City column. If you do this, you must leave the StateProvince column blank. The total length of the City and StateProvince columns cannot exceed 100 characters.

columnName	Description / Valid Values
StateProvince	The state or province of the city whose postal codes you want to look up. You may also put the state in the City column instead of the StateProvince column. The total length of the City and StateProvince columns cannot exceed 100 characters.
Country	The country code or name of the city whose postal codes you want to look up. The only valid value is US.

Options

Table 239: GetPostalCodes Options

optionName	Description/Valid Values
Database.US	Specifies the database to be used for postal code look-ups. Only databases that have been defined in the US Database Resources panel in the Management Console are available.
IncludeVanityCity	Specifies whether or not to include postal codes for the city's non-mailing city names. A non-mailing city name is an alternate name for the primary city name. For example, Hollywood is a non-mailing city name for Los Angeles. **Y** Yes, include postal codes for non-mailing city names. **N** No, do not include postal codes for non-mailing city names (default).
OutputCityType	Specifies whether or not to return the city type in the output. If enabled, the city type is returned in the City.Type column. **Y** Yes, include the city type in the output. **N** No, do not include the city type in the output (default).

Output

GetPostalCodes returns the postal codes for a specified city. Each postal code is returned in a separate record along with the data listed in the following table.

Note: The output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 240: GetPostalCodes Output

columnName	Description / Valid Values
City	The postal code's city.
City.Type	The USPS® city type (U.S. addresses only). The city type is determined by looking at the ZIP Code and the city name. For example, the city Lanham MD has the postal codes 20703, 20706, and 20784. Lanham is the primary city in 20703 and 20706 but is a vanity city in 20784. This field column is only populated if OutputCityType=Y. The possible values are: **V** Vanity (non-mailing) city name. **P** Primary. The city name is the primary mailing city name. **S** Secondary. The city name is an alternate city name but is acceptable. A city can have multiple secondary city names.
Country	The country code or name. The only valid value is US.
PostalCode	A postal code in the specified city.
ProcessedBy	Because this service only works for U.S. addresses, ProcessedBy will always contain one value: USA.
StateProvince	The postal code's state or province.
Status	Reports the success or failure of the match attempt. **null** Success **F** Failure
Status.Code	Reason for failure, if there is one. One of the following: • CountryNotSupported • UnableToLookup
Status.Description	Description of failure. • Input country is not supported

columnName	Description / Valid Values
	• Input city was blank • Input city & state / province was blank, or no match found • City-state mismatch (different spelling found, or city-state was a vanity name and vanity matching was not allowed, or city-state did not match ZIP Code)

GetPostalCodes Loqate

GetPostalCodesLoqate allows you to look up the postal codes for a particular city. The service takes a city, state, and country as input and returns the postal codes for that city. The input must be exactly correct in order to return postal codes.

GetPostalCodesLoqate is part of the Universal Addressing Module. For more information on this module, including a listing of other components included with it, see **What Is the Universal Addressing Module?** on page 722

Input

GetPostalCodesLoqate takes a city, state/province, and country as input.

Table 241: GetPostalCodesLoqate Input

columnName	Description / Valid Values
City	The city whose postal codes you want to look up. You may put the city and state in the City column. If you do this, you must leave the StateProvince column blank.
Country	The country code or name, in any of the following formats: • 2-digit ISO country code • 3-digit UPU Country code • English country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
StateProvince	The state or province of the city whose postal codes you want to look up. You may also put the state in the City column instead of the StateProvince column.

Options

Table 242: GetPostalCodesLoqate Options

optionName	Description/Valid Values	
Database	Specifies the database to be used for postal code look-ups. Only databases that have been defined in the Management Console are available.	
Data license error handling	Specifies how you want Spectrum Technology Platform to respond when a data license error occurs.	
	Fail the job	Fail the entire job if a data license error occurs.
	Fail the record	Fail the record(s) for which the data license error occurs and continue processing.

Output

GetPostalCodesLoqate returns the postal codes for a specified city. Each postal code is returned in a separate record along with the data listed in the following table.

Table 243: GetPostalCodesLoqate Output

columnName	Description / Valid Values	
PostalCode	A postal code in the specified city.	
ProcessedBy	Indicates which address coder processed the address.	
	LOQATE	The Loqate coder processed the address.
Status	Reports the success or failure of the match attempt.	
	null	Success
	F	Failure
Status.Code	Reason for failure, if there is one. One of the following: • InvalidCountry • UnableToLookup	

columnName	Description / Valid Values
Status.Description	Description of failure. • Input country is not supported • Input city was blank • Input city & state / province was blank, or no match found

ValidateAddress

ValidateAddress standardizes and validates addresses using postal authority address data. ValidateAddress can correct information and format the address using the format preferred by the applicable postal authority. It also adds missing postal information, such as postal codes, city names, state/province names, and more.

ValidateAddress also returns result indicators about validation attempts, such as whether or not ValidateAddress validated the address, the level of confidence in the returned address, the reason for failure if the address could not be validated, and more.

During address matching and standardization, ValidateAddress separates address lines into components and compares them to the contents of the Universal Addressing Module databases. If a match is found, the input address is *standardized* to the database information. If no database match is found, ValidateAddress optionally *formats* the input addresses. The formatting process attempts to structure the address lines according to the conventions of the appropriate postal authority.

ValidateAddress is part of the Universal Addressing Module. For more information on this module, including a listing of other components included with it, see **What Is the Universal Addressing Module?** on page 722.

Input

ValidateAddress takes an address as input. All addresses use this format regardless of the address's country. See **Address Line Processing for U.S. Addresses** on page 769 for important information about how address line data is processed for U.S. addresses.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 244: Input Format

columnName	Format	Description / Valid Values
AddressLine1	String [50]	The first address line.

columnName	Format	Description / Valid Values
AddressLine2	String [50]	The second address line.
AddressLine3	String [50]	The third address line. Does not apply to Canadian addresses.
AddressLine4	String [50]	The fourth address line. Does not apply to Canadian addresses.
AddressLine5	String [50]	The fifth address line. Applies only to U.K. addresses. May contain street name, unit number, building number, and so on.
City	String [50]	The city name. For U.S. addresses only, you may put the city, state, and ZIP Code™ in the City field. If you do this, you must leave the StateProvince and PostalCode fields blank.
StateProvince	String [50]	The state or province. For U.S. addresses only, you may put the state in the City field instead of the StateProvince field.
PostalCode	String [10]	The postal code for the address in one of the following formats: 99999 99999-9999 A9A9A9 A9A 9A9 9999 999 For U.S. addresses only, you may put the ZIP Code™ in the City field. For U.S. addresses only, if the city/state/ZIP Code™ is in the PostalCode field, ValidateAddress may parse the data and successfully process the address. For best results, put this data in the appropriate fields (City, StateProvince, and PostalCode).
Country	String [50]	The country code or name, in any of the following formats: • Two-character ISO 3116-1 Alpha-2 country code

columnName	Format	Description / Valid Values
		• Three-character ISO 3116-1 Alpha-3 country code • English country name • French country name • German country name • Spanish country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	String [50]	The company or firm name.
USUrbanName	String [50]	The U.S. address urbanization name. This is used primarily for Puerto Rico addresses.
CustomerID	String [9]	If this mailpiece uses a generic barcode, specify your USPS®-assigned customer ID in this field. The ValidateAddress generic barcode is used for mailpieces that use the OneCode ACS® service.
CanLanguage	String	For Canadian addresses only, indicates whether the address is in English or French, if the option CanFrenchFormat=T is used. If this field is blank, the address is formatted in English. If the field contains any non-blank value, the address is formatted in French. Note that addresses in Quebec are always formatted in French regardless of the value in this field.

Address Line Processing for U.S. Addresses

The input fields AddressLine1 through AddressLine4 are handled differently for U.S. addresses depending on whether the firm name extraction or urbanization code extraction options are enabled. If either of these options is enabled, ValidateAddress will look at the data in all four fields to validate the address and extract the requested data (firm name and/or urbanization code). If neither of these options is enabled, ValidateAddress uses only the first two non-blank address line fields in its validation attempt. The data in the other address line fields is returned in the output field AdditionalInputData. For example,

AddressLine1: A1 Calle A
AddressLine2:
AddressLine3: URB Alamar
AddressLine4: Pitney Bowes Software

In this address, if either firm name extraction or urbanization code extraction were enabled, ValidateAddress would examine all four address lines. If neither firm name extraction nor urbanization code extraction were enabled, ValidateAddress would examine AddressLine1 and AddressLine3 (the

first two non-blank address lines) and attempt to validate the address using that data; the data in AddressLine4 would be returned in the output field AdditionalInputData.

Options

Output Data Options

The following table lists the options that control the type of information returned by ValidateAddress. Some of these options can be overridden for Canadian addresses. For more information, see **Canadian Address Options** on page 792.

Table 245: Output Data Options

optionName	Description/Valid Values
OutputRecordType	Type of output record. For more than one, provide a list. **A** Returns 1 to 4 lines of address data plus city, state, postal code, firm name, and urbanization name information. Each address line represents an actual line of the address as it would appear on an envelope. For more information, see **Output** on page 802. If ValidateAddress could validate the address, the address lines contain the standardized address. When addresses are standardized, punctuation is removed, directionals are abbreviated, street suffixes are abbreviated, and address elements are corrected. If ValidateAddress could not validate the address, the address lines contain the address as it appeared in the input ("pass through" data). Non-validated addresses are always included as pass through data in the address line fields even if you do not specify **OutputRecordType=A**. **E** Parsed address elements. Each part of the address, such as house number, street name, street suffix, directionals, and so on is returned in a separate field. For more information, see **Parsed Address Elements Output** on page 805. Note that if you specify "E" and specify **OutputFormattedOnFail=Y**, the parsed address elements will contain the input address for addresses that could not be validated. **I** Parsed input. This option returns the input address in parsed form regardless of whether or not ValidateAddress is able to validate the address. Each part of the input address, such as house number, street name, street suffix, directionals, and so on is

optionName	Description/Valid Values
	returned in a separate field. Parsed input (value "I") differs from the combination of **OutputRecordType=E** and **OutputFormattedOnFail=Y** in that "I" returns all input address in parsed form, not just input that could not be validated. For more information, see **Parsed Input** on page 807. **P** Postal data. Output addresses contain additional data for each validated address. For more information, see **Postal Data Output** on page 809. **Blank** Do not return any address data or postal data.
OutputFieldLevelReturnCodes	Specifies whether to include field-level result indicators. Field-level result indicators describe how ValidateAddress handled each address element. Field-level result indicators are returned in the qualifier "Result". For example, the field-level result indicator for HouseNumber is contained in **HouseNumber.Result**. For a complete listing of result indicator output fields, see **Field-Level Result Indicators** on page 815. **N** No, do not output field-level return codes (default). **Y** Yes, output field-level return codes.
OutputFormattedOnFail	Specifies whether to return a formatted address when an address cannot be validated. The address is formatted using the preferred address format for the address's country. If this option is not selected, the output address fields are blank when ValidateAddress cannot validate the address. **Note:** This option applies only to U.S. and Canadian addresses. Formatted data will not be returned for any other address. **N** No, do not format failed addresses (default). **Y** Yes, format failed addresses. Formatted addresses are returned using the format specified by the**OutputRecordType** option. Note that if you specify **OutputRecordType=E**, the parsed address elements will contain the parsed, validated address for addresses that could be validated. If the address could not be validated the parsed address elements will contain the input address in parsed form. If you always want the output to contain the input

optionName	Description/Valid Values
	address in parsed form, regardless of whether or not ValidateAddress could validate the address, specify **OutputRecordType=I**. If you specify Y, you must specify "A" and/or "E" for OutputRecordType.
OutputStreetNameAlias	For U.S. addresses only, specifies whether or not to use a street's alias in the output. A street alias is an alternate name for a street and typically applies only to a specific range of addresses on the street. If you do not allow street aliases in the output then the street's "base" name will appear in the output regardless of whether or not there is an alias for the street. The base name is the name that applies to the entire street. **N** No, do not return street name aliases in the output. **Y** Yes, return street name aliases in the output if there is an alias for the street (default).
OutputStreetNameAlias	For U.S. addresses only, specifies how to handle street name aliases used in the input. A street alias is an alternate name for a street and typically applies only to a specific range of addresses on the street. If you enable this option, street name aliases used in the input will appear in the output. If you do not enable this option, street name aliases in the input will be converted to the base street name in the output, with the following exceptions: • If a preferred alias is used in input the preferred alias will always be used in output. • Changed aliases used in input are always converted to the base street name in output. This is one of three options that control how ValidateAddress handles street name aliases. The other two are OutputPreferredAlias and OutputAbbreviatedAlias. For information on these options, see **U.S. Address Options** on page 783. **Note:** If OutputAbbreviatedAlias is enabled, the abbreviated alias will always appear in the output even if you have OutputStreetNameAlias disabled. One of the following:

optionName	Description/Valid Values
	N No, do not return street name aliases in the output. **Y** Yes, return street name aliases in the output if the input street name is an alias (default).
OutputAddressBlocks	Specifies whether to return a formatted version of the address as it would be printed on a physical mailpiece. Each line of the address is returned in a separate address block field. There can be up to nine address block output fields: **AddressBlock1** through **AddressBlock9**. For example, this input address: AddressLine1: 4200 Parliament Place AddressLine2: Suite 600 City: Lanham StateProvince: MD PostalCode: 20706 Results in this address block output: AddressBlock1: 4200 PARLIAMENT PL STE 600 AddressBlock2: LANHAM MD 20706-1882 AddressBlock3: UNITED STATES OF AMERICA ValidateAddress formats the address into address blocks using postal authority standards. The country name is returned using the Universal Postal Union country name. Note that the option OutputCountryFormat does not affect the country name in the address block, it only affects the name returned in the **Country** output field. For addresses outside the U.S. and Canada, if ValidateAddress is unable to validate the address, no address blocks are returned. For addresses in the U.S. and Canada, address blocks are returned even if validation fails. One of the following: **N** No, do not return address blocks. Default. **Y** Yes, return address blocks.

Obtaining Congressional Districts

ValidateAddress can determine the U.S. congressional district for an address.

Option

To obtain congressional districts, `OutputRecordType` must contain P. For more information on `OutputRecordType`, see **Output Data Options** on page 770.

Output

The following output column contains the congressional district when you specify `OutputRecordType=P`.

Table 246: Congressional District Output

columnName	Description/Valid Values
USCongressionalDistrict	Congressional district number. If the address is a non-state address (for example Puerto Rico or Washington D.C.) this field is blank.

Obtaining County Names

ValidateAddress can determine the county where a particular address is located and return the county name.

Note: County names are available for U.S. addresses only.

Option

To obtain county names, `OutputRecordType` must contain P. For more information on `OutputRecordType`, see **Output Data Options** on page 770.

Output

The following output fieldcolumn contains the county name when you specify **OutputRecordTyp**e=P.

Table 247: County Name Output

columnName	Description/Valid Values
USCountyName	County name

Obtaining FIPS County Numbers

Federal Information Processing Standards (FIPS) county numbers are numbers that identify each county in a state. Note that these numbers are only unique at the state level, not the national level. For more information, see **http://www.census.gov**.

Note: FIPS county numbers are available for U.S. addresses only.

Option

To obtain FIPS county numbers, `OutputRecordType` must contain P. For more information on `OutputRecordType`, see **Output Data Options** on page 770.

Output

The following output column contains the FIPS county number when you specify `OutputRecordType=P`.

Table 248: FIPS County Number Output

columnName	Description/Valid Values
USFIPSCountyNumber	FIPS (Federal Information Processing Standards) county number

Obtaining Carrier Route Codes

Carrier route codes are unique identifiers assigned to each mail carrier who delivers mail, allowing unique identification of each U.S. delivery route. ValidateAddress can return the code that represents an addressee's carrier route.

Note: Carrier route codes are available for U.S. addresses only.

Option

To obtain carrier route codes, `OutputRecordType` must contain P. For more information on `OutputRecordType`, see **Output Data Options** on page 770.

Output

The following output field contains the carrier route code when you select specify `OutputRecordType=P`.

Table 249: Carrier Route Code Output

Field Name columnName	Description/Valid Values
USCarrierRouteCode	Carrier route code

Creating Delivery Point Barcodes

A Delivery Point Barcode (DPBC) is a POSTNET™ barcode representation of the address. It consists of 62 bars with beginning and ending frame bars and five bars each for the ZIP + 4® Code, a value calculated based on the street address number, and a correction digit. The DPBC allows automated sortation of letter mail to the carrier level in walk sequence. ValidateAddress generates the data you need to assemble a DPBC.

Note: Delivery Point Barcodes are available for U.S. addresses only. For more information on Delivery Point Barcodes, see **http://www.usps.com**.

Option

To generate the data needed to assemble a DPBC, `OutputRecordType` must contain P. The following table lists the possible values for `OutputRecordType`. For more information on `OutputRecordType`, see **Output Data Options** on page 770.

Output

The following output contains DPBC information when you specify `OutputRecordType=P`.

Table 250: Delivery Point Barcode Output

columnName	Description/Valid Values
PostalBarCode	The delivery point portion of the delivery point barcode.
USBCCheckDigit	Check-digit portion of the 11-digit delivery point barcode.

Assembling a DPBC

To assemble a DPBC you concatenate the values found in the ValidateAddress output columns as follows:

PostalCode.Base + PostalCode.Addon + PostalBarcode + USBCCheckDigit

For example, if you have the following:

- **PostalCode.Base** = 49423
- **PostalCode.Addon** = 4506
- **PostalBarcode** = 29
- **USBCCheckDigit** = 2

The assembled barcode would be:

494234506292

Default Options

The following table lists the options that control the format and processing of addresses. These are called "default options" because by default the apply to all addresses. Some of these options can be overridden for Canadian addresses. For more information, see **Canadian Address Options** on page 792.

Table 251: Default Options

optionName	Description/Valid Values
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
OutputPostalCodeSeparator	Specifies whether to use separators (spaces or hyphens) in ZIP™ Codes or Canadian postal codes. For example, a ZIP + 4® Code with the separator would be 20706-1844 and without the separator it would be 207061844. A Canadian postal code with the separator would be P5E"1S7 and without the separator it would be P5E1S7. **Y** Yes, use separator (default). **N** No, do not use separator. **Note:** Spaces are used in Canadian postal codes and hyphens in U.S. ZIP + 4® Codes.
OutputMultinationalCharacters	Specifies whether or not to return multinational characters, including diacritical marks such as umlauts or accents. (Not supported for U.S. addresses). **N** No, do not use multinational characters in the output (default). Only standard ASCII characters is returned. **Y** Yes, use multinational characters in the output.
KeepMultimatch	Indicates whether or not to return multiple address for those input addresses that have more than one possible match. **Y** Yes, return multiple matches (default). **N** No, do not return multiple matches. For more information, see **Returning Multiple Matches** on page 782.

optionName	Description/Valid Values
StandardAddressFormat	Specifies where to place secondary address information for U.S. addresses. Secondary address information refers to apartment numbers, suite numbers, and similar designators. For example, in this address the secondary address information is "Apt 10E" and the primary address information is "424 Washington Blvd". Apt 10E 424 Washington Blvd Springfield MI 49423 **C** Place both primary and secondary address information in AddressLine1 (default). **S** Place the primary address information in AddressLine1 and the secondary address information in AddressLine2. **D** Place both primary and secondary address information in AddressLine1 and place dropped information from dual addresses in AddressLine2. A dual address is an address that contains both street information and PO Box/Rural Route/Highway Contract information. For more information, see **About Dual Address Logic** on page 781.
OutputShortCityName	Specifies how to format city names that have short city name or non-mailing city name alternatives. Applies to U.S. and Canadian addresses. **Y** Returns the USPS®-approved abbreviation for the city, if there is one. The USPS® provides abbreviations for city names that are 14 characters long or longer. City abbreviations are 13 characters or less and can be used when there is limited space on the mailing label. If there is no short city name for the city, then the full city name is returned. **N** Returns the long city name (default). **S** Returns the abbreviated city name only if an abbreviated city name is used in the input address. If the input address does not use a short city name, either the long or short city name could be returned, depending on USPS® regulations for the particular city. Select this option if you are performing a CASS™ test. **V** Output the non-mailing city name (the vanity name) if the input city name is a non-mailing city name. For example, "Hollywood" is a non-mailing city name for

optionName	Description/Valid Values
	"Los Angeles". If you do not select this option and the input city name is a non-mailing city name the long version of the mailing city is returned.
OutputCountryFormat	Specifies the format to use for the country name returned in the **Country** output field. For example, if you select English, the country name "Deutschland" would be returned as "Germany". **E** Use English country names (default). **S** Use Spanish country names. **F** Use French country names. **G** Use German country names. **I** Use two-letter ISO abbreviation for the countries instead of country names. **U** Use Universal Postal Union abbreviation for the countries instead of country names.
HomeCountry	Specifies the default country. You should specify the country where most of your addresses reside. For example, if most of the addresses you process are in Canada, specify Canada. ValidateAddress uses the country you specify to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields. The valid country names are: Afghanistan, Albania, Algeria, American Somoa, Andorra, Angola, Anguilla, Antigua And Barbuda, Argentina, Armenia, Aruba, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bermuda, Bhutan, Bolivia, Bosnia And Herzegovina, Botswana, Brazil, British Virgin Islands, Brunei Darussalam, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon, Canada, Cape Verde, Cayman Islands, Central African Republic, Chad, Chile, China, Colombia, Comoros Islands, Congo, Cook Islands, Costa Rica, Cote D'Ivoire, Croatia, Cuba, Cyprus, Czech Republic, Democratic Republic Of Congo, Denmark, Djibouti, Dominica, Dominican Republic, East Timor, Ecuador, Egypt, El Salvador, Equitorial Guinea, Eritrea, Estonia, Ethiopia, Falkland Islands, Faroe Islands, Federated States Of Micronesia, Fiji, Finland, France, French Guiana, Gabon, Gambia, Germany, Ghana, Gibraltar, Greece,

optionName	Description/Valid Values
	Greenland, Grenada, Guadeloupe, Guam, Guatemala, Guinea, Guinea Bissau, Guyana, Haiti, Holy See, Honduras, Hong Kong, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Korea, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania, Luxembourg, Macau, Macedonia, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall Islands, Martinique, Mauritania, Mauritius, Mayotte, Mexico, Moldova, Monaco, Mongolia, Monserrat, Morocco, Mozambique, Myanmar, Namibia, Nauru, Nepal, Netherlands Antilles, New Caledonia, New Zealand, Nicaragua, Niger, Nigeria, Niue, Norway, Oman, Pakistan, Palau, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Pitcairn Islands, Poland, Portugal, Puerto Rico, Qatar, Republic Of Georgia, Republic Of Korea, Republic Of Singapore, Reunion, Romania, Russia, Rwanda, Saint Helena, Saint Kitts And Nevis, Saint Lucia, Saint Pierre And Miquelon, Saint Vincent And The Grenadines, Samoa, San Marino, Sao Tome And Principe, Saudi Arabia, Senegal, Seychelles, Sierra Leone, Slovakia, Slovenia, Solomon Islands, Somalia, South Africa, Spain, Sri Lanka, Sudan, Surivalue, Swaziland, Sweden, Switzerland, Syria, Tahiti, Taiwan, Tajikistan, Tanzania, Thailand, The Netherlands, Togo, Tonga, Trinidad And Tobago, Tristan Da Cunha, Tunisia, Turkey, Turkmenistan, Turks And Caicos Islands, Tuvalu, Uganda, Ukraine, United Arab Emirates, United Kingdom, United States, Uruguay, Uzbekistan, Vanuatu, Venezuela, Vietnam, Virgin Islands (US), Wallis And Futuna, Yemen, Yugoslavia, Zambia, Zimbabwe
DualAddressLogic	Indicates how ValidateAddress should return a match if multiple non-blank address lines are present or multiple address types are on the same address line (U.S. addresses only). **N** (Default) USPS® CASS™ regulations determine the address returned based on the following order of priority: **1.** PO Box **2.** Firm **3.** Highrise **4.** Street **5.** Rural Route **6.** General Delivery

optionName	Description/Valid Values
	S Return a street match, regardless of the address line. **P** Return a PO Box match, regardless of the address line. For more information, see **About Dual Address Logic** on page 781.

About Dual Address Logic

For U.S. addresses only, the DualAddressLogic option controls whether ValidateAddress should return a street match or a PO Box/Rural Route/Highway Contract match when the address contains both street and PO Box/Rural Route/Highway Contract information in the same address line.

Note: The DualAddressLogic option has no effect if the street information is in a different address line input field than the PO Box/Rural Route/Highway Contract information.

For example, given the following input address:

AddressLine1: 401 N Main St Apt 1 POB 1
City: Kemp
StateProvince: TX
PostalCode: 75143

ValidateAddress would return one of the following:

- If DualAddressLogic is set to either N or P, ValidateAddress returns the following:

  AddressLine1: PO Box 1
  City: Kemp
  StateProvince: TX
  PostalCode: 75143-0001

- If DualAddressLogic is set to S, ValidateAddress returns the following:

  AddressLine1: 401 N Main St Apt 1
  City: Kemp
  StateProvince: TX
  PostalCode: 75143-4806

The address data that is not used to standardize the address can be returned in one of two places:

- **AddressLine2**—The address information not used to standardize the address is returned in the **AddressLine2** field if you specify StandardAddressFormat=D. For more information on this option, see **Default Options** on page 776. For example, if you choose to return a street match for dual addresses,

  AddressLine1: 401 N Main St Apt 1
  AddressLine2: PO Box 1
  City: Kemp
  StateProvince: TX
  PostalCode: 75143-0001

- **AdditionalInputData**—If you do not specify StandardAddressFormat=D then the address information not used to standardize the address is returned in the **AdditionalInputData** field. For more information on this option, see **Default Options** on page 776. For example, if you choose to return a street match for dual addresses,

  AddressLine1: 401 N Main St Apt 1
  City: Kemp
  StateProvince: TX
  PostalCode: 75143-0001
  AdditionalInputData: PO Box 1

Address information that is dropped can be retrieved by setting the StandardAddressFormat option to D. For more information, see **Default Options** on page 776 .

Returning Multiple Matches

If ValidateAddress finds multiple address in the postal database that are possible matches for the input address, you can have ValidateAddress return the possible matches. For example, the following address matches multiple addresses in the U.S. postal database:

PO BOX 1 New York, NY

Options

To return multiple matches, use the options described in the following table.

Table 252: Multiple Match Option

optionName	Description/Valid Values
KeepMultimatch	Indicates whether or not to return multiple address for those input addresses that have more than one possible match. **Y** Yes, return multiple matches (default). **N** No, do not return multiple matches.
MaximumResults	A number between 1 and 10 that indicates the maximum number of addresses to return. The default value is 1. **Note:** The difference between Keepmultimatch=N and KeepMultimatch=Y/MaximumResults=1 is that a multiple match will return a failure if KeepMultimatch=N, whereas a multiple match will return one record if KeepMultimatch=Y and MaximumResults=1.
OutputFieldLevelReturnCodes	To identify which output addresses are candidate addresses, you must specify a value of **Y** for OutputFieldLevelReturnCodes. When you do this, records that are candidate addresses will have one or more "M" values in the field-level result indicators.

Output

When you choose to return multiple matches, the addresses are returned in the address format you specify. For information on specifying address format, see **Output Data Options** on page 770. To identify which records are the candidate addresses, look for multiple "M" values in the field-level result indicators. For more information, see **Field-Level Result Indicators** on page 815.

U.S. Address Options

optionName	Description/Valid Values
PerformUSProcessing	Specifies whether to process U.S. addresses. If you enable U.S. address processing ValidateAddress will attempt to validate U.S. addresses. If you disable U.S. address processing, U.S. addresses will fail, meaning they are returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for U.S. address processing you must disable U.S. address processing in order for your jobs to complete successfully, regardless of whether or not they contain U.S. addresses. **Note:** You must have a valid license for U.S. address processing to successfully process U.S. addresses. If you enable U.S. address processing but are not licensed for this feature, or your license has expired, you will receive an error. One of the following: **N** No, do not process U.S. addresses. **Y** Yes, process U.S. addresses. Default.
Database.US	Specifies which database to use for validating U.S. addresses. Only databases that have been defined in the US Database Resources panel in the Management Console are available.
PerformLOT	Enhanced Line of Travel (eLOT) processing assigns a Line of Travel sequence code to your addresses. Note that ValidateAddress does not sort into eLOT sequence but it provides data (the Line of Travel sequence code) that allows you to sort addresses into eLOT sequence. To perform eLOT processing you must have the eLOT database installed. For more information, see **Universal Addressing Databases** on page 722. One of the following: **N** No, do not perform Line of Travel Processing. Default. **Y** Yes, perform Line of Travel processing.

optionName	Description/Valid Values
	For a listing of the output fields returned by this option, see **Enhanced Line of Travel Output** on page 826.
PerformRDI	Residential Delivery Indicator (RDI™) processing checks if an address is a residential address (not a business address). To perform RDI™ processing, you must have the RDI™ database installed. If you enable both DPV® and RDI™ processing, RDI™ information is only returned if the address is a valid delivery point. If DPV® does not validate the address no RDI™ data is returned. One of the following: **N** No, do not perform Residential Delivery Indicator processing. Default. **Y** Yes, perform Residential Delivery Indicator processing.
PerformESM	Enhanced Street Matching (ESM) applies additional matching logic to correct misspelled or complex street names and obtain a match. ESM enables ValidateAddress to validate more addresses but it reduces performance. You cannot perform ESM when ASM is enabled. One of the following: **N** No, do not perform enhanced street matching. Default. **Y** Yes, perform enhanced street matching.
PerformASM	All Street Matching (ASM) applies ESM processing as well as additional matching logic to correct errors in street names and obtain a match. It is effective at matching streets when the first letter of the street is incorrect. ASM provides the best address validation but reduces performance. One of the following: **N** No, do not perform all street matching. **Y** Yes, perform all street matching. Default.
PerformDPV	Delivery Point Validation (DPV®) validates that a specific address exists, as opposed to validating that a specific address is within a range of valid addresses. CMRA processing checks if an address is for a mailbox rented from a private company, referred to as a Commercial Mail Receiving Agent (CMRA).

optionName	Description/Valid Values
	To perform DPV and CMRA processing, you must have the DPV database installed. The DPV database contains both DPV and CMRA data. For more information, see **Universal Addressing Databases** on page 722. One of the following: **N** No, do not perform Delivery Point Validation or CMRA processing. Default. **Y** Yes, perform Delivery Point Validation and CMRA processing. For a listing of the output fields returned by this option, see **DPV and CMRA Output** on page 828. For information about false-positive matches, which can disable your DPV key, see **Reporting DPV False-Positive Violations** on page 886.
PerformLACSLink	The USPS® Locatable Address Conversion System (LACS) allows you to correct addresses that have changed as a result of a rural route address converting to street-style address, a PO Box renumbering, or a street-style address changing. When enabled, LACS^{Link} processing is attempted for addresses that could not be validated, or addresses were validated and flagged for LACS^{Link} conversion. To perform LACS^{Link} processing, you must have the LACS^{Link} database installed. For more information, see **Universal Addressing Databases** on page 722. One of the following: **N** No, do not attempt LACS^{Link} conversion. Default. **Y** Yes, attempt LACS^{Link} conversion. For a listing of the output fields returned by this option, see **LACSLink Output** on page 826 For information about false-positive matches, which can disable your LACS^{Link} key, see **Reporting LACS/Link False-Positive Violations** on page 889.
PerformEWS	The Early Warning System (EWS) uses the USPS® EWS File to validate addresses that are not in the ZIP + 4® database. To perform EWS processing, you must have the EWS database installed. For more information, see **Universal Addressing Databases** on page 722. If an input address matches an address in the EWS file, ValidateAddress will return the following record-level result indicators:

optionName	Description/Valid Values
	• Status="F" • Status.Code="EWSFailure" • Status.Description="Address found in EWS table" One of the following: **N** No, do not perform EWS processing. Default. **Y** Yes, perform EWS processing.
ExtractFirm	Specifies whether to extract the firm name from AddressLine1 through AddressLine4 and place it in the FirmName output field. This option works in cases where the input record's FirmName field is blank and there is more than one address line. One of the following: **Y** Yes, extract the firm name. **N** No, do not extract the firm name. Default. To identify firm names in address lines, ValidateAddress scans the address lines for keywords and patterns that help it identify which fields are address lines and which are FirmName lines. Since this is done based on patterns, ValidateAddress may misidentify fields. The following tips can help ensure optimal firm extraction: • If possible, place the primary address elements in AddressLine1, the secondary elements in AddressLine2, Urbanization in AddressLine3, and firm in AddressLine4. If the address has no urbanization code, then place the firm name in AddressLine3 and leave AddressLine4 blank. For example, **AddressLine1:** 4200 Parliament Place **AddressLine2:** Suite 600 **AddressLine3:** Pitney Bowes Software **AddressLine4:** <blank> • When you define just two address lines, AddressLine2 is assigned to the secondary address most of the time. If you want to increase the chance that ValidateAddress will treat AddressLine2 as a firm name, put the firm name in AddressLine3 and leave AddressLine2 blank. • Numerics in a firm name (such as the "1" in "1 Stop Software") will increase the likelihood that ValidateAddress will treat the field as an address line. Here are some examples of firm name extraction:

optionName	Description/Valid Values
	• In this example, AddressLine2 would get extracted into the FirmName output field **FirmName:** <blank> **AddressLine1:** 4200 Parliament Place Suite 600 **AddressLine2:** International Goose Feathers inc. • In this example, AddressLine3 would get extracted into the FirmName output field. **FirmName:** <blank> **AddressLine1:** 4200 Parliament Place **AddressLine2:** Suite 600 **AddressLine3:** Pitney Bowes Software • In this example, AddressLine3 would be placed in the AdditionalInputData output field. The firm name would not be extracted because the FirmName input field is not blank. **FirmName:** International Goose Feathers Inc. **AddressLine1:** 4200 Parliament Place **AddressLine2:** Suite 600 **AddressLine3:** Pitney Bowes Software • In this example, no firm name would be extracted because there is only one non-blank address line, which is always treated as the primary address element. **FirmName:** <blank> **AddressLine1:** 4200 Parliament Place Suite 600 • In this example, AddressLine2 would be treated as a secondary address element because the numeral "1" causes ValidateAddress to treat that field as a secondary address element. **FirmName:** <blank> **AddressLine1:** 4200 Parliament Place Suite 600 **AddressLine2:** Pitney Bowes Software
ExtractUrb	Specifies whether to extract the urbanization name from AddressLine1 through AddressLine4 and place it in the USUrbanName output field. This option works in cases where the input record's USUrbanName field is blank and there is more than one address line. One of the following: **Y** Yes, extract the urbanization name.

optionName	Description/Valid Values
	N No, do not extract the urbanization name. Default. To identify urbanization names, ValidateAddress scans the address lines for keywords and patterns that help it identify which fields are address lines and which are urbanization name lines. Since this is done based on patterns, it is possible for ValidateAddress to incorrectly identify fields. To help ensure optimal urbanization extraction, place the primary address elements in AddressLine1, the secondary elements in AddressLine2, Urbanization in AddressLine3, and firm in AddressLine4, if possible. For example, **AddressLine1:** A1 Calle A **AddressLine2:** **AddressLine3:** URB Alamar **AddressLine4:** Pitney Bowes Software
PerformSuiteLink	Specifies whether to perform SuiteLink™ processing. SuiteLink™ corrects secondary address information (for example, a suite number) for business addresses. For more information, see **U.S. Addressing Concepts** on page 726. To perform SuiteLink™ processing, you must have the SuiteLink™ database installed. One of the following: **N** No, do not use SuiteLink™. Default. **Y** Yes, use SuiteLink™ processing. For a listing of fields returned by this option, see **SuiteLink Output** on page 829.
OutputPreferredAlias	Specifies whether to use a street's preferred alias in the output. If the preferred alias is used in the input then the preferred alias will be the street name in the output regardless of whether you enable this option. For information about street aliases, see **Street Name Aliases** on page 730. This is one of three options that control how ValidateAddress handles street name aliases. The other two are OutputStreetNameAlias and OutputAbbreviatedAlias. In most cases, if you select both OutputPreferredAlias and OutputAbbreviatedAlias , and ValidateAddress finds both a preferred and an abbreviated alias in the postal database, the abbreviated alias will be used in the output. The exception to this rule is if the input street

optionName	Description/Valid Values
	name is a preferred alias. In this case, the preferred alias will be used in the output. One of the following: **Y** Yes, perform preferred alias processing. **N** No, do not perform preferred alias processing. Default. **Note:** If the input address contains a street name alias of type "changed" the output address will always contain the base street name regardless of the options you specify.
OutputAbbreviatedAlias	Specifies whether to use a street's abbreviated alias in the output if the output address line is longer than 31 characters. For information about street aliases, see **Street Name Aliases** on page 730. This is one of three options that control how ValidateAddress handles street name aliases. The other two are OutputStreetNameAlias and OutputPreferredAlias. **Note:** If a preferred alias is specified in the input, the output street name will always be the preferred alias, even if you enable abbreviated street name alias processing. One of the following: **Y** Yes, perform abbreviated alias processing. **N** No, do not perform abbreviated alias processing. Default. **Note:** If the input address contains a street name alias of type "changed" the output address will always contain the base street name regardless of the options you specify.
DPVDetermineNoStat	Determines the "no stat" status of an address. An address is considered "no stat" if it exists but cannot receive mail, and therefore is not counted as a delivery statistic on a carrier's route (hence the term "no stat"). Examples include buildings under construction or those that the letter carrier has identified as not likely to receive mail. One of the following: **N** No, do not determine "no stat" status. Default. **Y** Yes, determine "no stat" status. **Note:** You must enable DPV processing to use this option.

optionName	Description/Valid Values
	The result is returned in the DPVNoStat field. For more information see **LACSLink Output** on page 826
DPVDetermineVacancy	Determines if the location has been unoccupied for at least 90 days. One of the following: **N** No, do not determine vacancy. Default. **Y** Yes, determine vacancy. **Note:** You must enable DPV processing to use this option. The result is returned in the DPVVacant field. For more information see **LACSLink Output** on page 826
ReturnVerimove	Returns VeriMove detail data in output. One of the following: **N** No, do not return VeriMove detail data. Default. **Y** Yes, return VeriMove detail data.
StreetMatchingStrictness	Specifies the algorithm to use when determining if an input address matches an address in the postal database. One of the following: **E** The input street name must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium" (default). **L** The matching algorithm is "loose."
FirmMatchingStrictness	Specifies the algorithm to use when determining if an input address matches an address in the postal database. One of the following: **E** The input firm name must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium" (default). **L** The matching algorithm is "loose."
DirectionalMatchingStrictness	Specifies the algorithm to use when determining if an input address matches an address in the postal database. One of the following:

optionName	Description/Valid Values
	E The input directionals, such as the "N" in 123 N Main St., must match the database exactly. **T** The matching algorithm is "tight." **M** The matching algorithm is "medium". Default. **L** The matching algorithm is "loose."
DPVSuccessfulStatusCondition	Select the match condition where a DPV result does NOT cause a record to fail. **F** Full match **P** Partial match **A** Always. Default. **Note:** Requires PerformDPV set to (**Y**).
FailOnCMRAMatch	Treat Commercial Mail Receiving Agency (CMRA) matches as failures? **N** No, do not treat CMRA matches as failures. Default. **Y** Yes, treat CMRA matches as failures. **Note:** Requires PerformDPV set to (**Y**).
StandardAddressPMBLine	Specifies where ValidateAddress places Private Mailbox (PMB) information. **N** None. Do not include the PMB information in Standard Address output (default). **1** Place the PMB information in AddressLine1. If you specify 1, you must set StandardAddressFormat to either C or D. **2** Place the PMB information in AddressLine2.

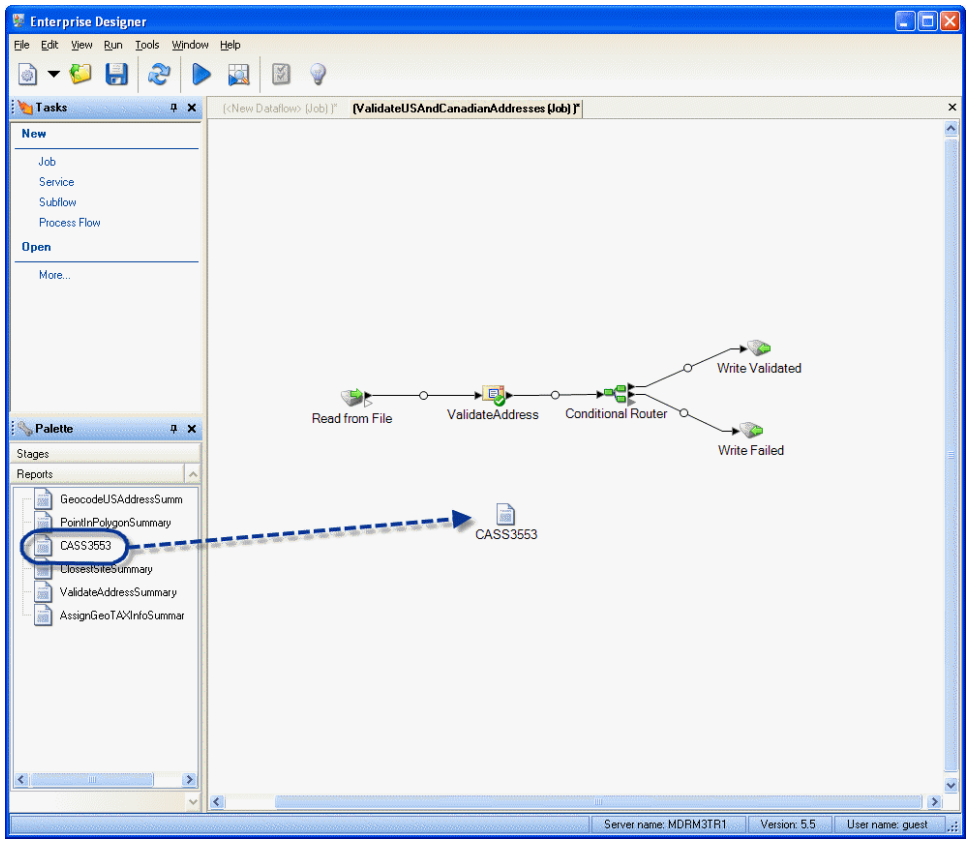
CASS Certified Processing

1. Validate Address must be in CASS Certified™ mode. If **(Not CASS Certified)** appears at the top of the window, click the **Enable CASS** button. The **Enforce CASS rules** check box will appear.
2. Click **Configure CASS 3553**. The **CASS Report Fields** dialog box appears.
3. Type the **List Processor** company name, **List Name or ID#**, and the **Number of Lists** being processed for this job.
4. Type the **Mailer Name, Address**, and **City, State, ZIP**.

5. Click **OK**.

   The List information will appear in Section B and the Mailer information in Section D of the generated USPS® CASS Form 3553.

6. In Enterprise Designer, drag the **CASS3553** report from the Reports pallet to the canvas.



7. Double-click the **CASS3553** icon on the canvas.
8. On the **Stages** tab, check the **Validate Address** checkbox. Note that if you have renamed the Validate Address stage to something else, you should check the box with the name you have given the address validation stage.
9. On the **Parameters** tab, select the format for the report. You can create the report in PDF, HTML, or plain text format.
10. Click **OK**.

Canadian Address Options

optionName	Description/Valid Values
PerformCanadianProcessing	Specifies whether to process Canadian addresses. If you enable Canadian address processing ValidateAddress will attempt to validate Canadian

optionName	Description/Valid Values
	addresses. If you disable Canadian address processing, Canadian addresses will fail, meaning they is returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for Canadian address processing you must disable Canadian address processing in order for your jobs to complete successfully, regardless of whether or not they contain Canadian addresses. **Note:** You must have a valid license for Canadian address processing to successfully process Canadian addresses. If you enable Canadian address processing but are not licensed for this feature, or your license has expired, you will receive an error. One of the following: **N** No, do not process Canadian addresses. **Y** Yes, process Canadian addresses (default).
Database.Canada	Specifies which database you want to use for validating Canadian addresses. To specify a database for Canadian address validation, select a database in the **Database** drop-down list. Only databases that have been defined in the **CAN Database Resources** panel in the Management Console are available.
CanFrenchFormat	Specifies how to determine the language (English or French) to use to format the address and directional. The following example shows an address formatted in English and French: English: 123 Main St W French: 123 Rue Main O The parameter controls the formatting of the address. It also affects the spelling of the directional but not spelling of the suffix. **C** Use the street suffix returned by the matching process to determine the language. The street suffix returned by the matching process, which is used internally by ValidateAddress during

optionName	Description/Valid Values
	processing, may be different from that in the input address. Ambiguous records are formatted like the input. Default. All addresses in Quebec are formatted using French. **S** Use the Canadian database to determine the language. The Canadian database contains data from the Canada Post Corporation (CPC). All addresses in Quebec are formatted using French. **T** Use the CanLanguage input field to determine the language. If there is a non-blank value in this field the address are formatted using French.
CanEnglishApartmentLabel	For English addresses, specifies the default apartment label to use in the output if there is no apartment label in the input address. This setting is ignored if you specify CanStandardAddressFormat=F. **Apt** Use "Apt" as the label. Default. **Apartment** Use "Apartment" as the label. **Suite** Use "Suite" as the label. **Unit** Use "Unit" as the label.
CanFrenchApartmentLabel	For French addresses, specifies the default apartment label to use in the output if there is no apartment label in the input address. This setting is ignored if you specify CanStandardAddressFormat=F. **App** Use "App" as the label. Default. **Appartement** Use "Appartement" as the label. **Bureau** Use "Bureau" as the label. **Suite** Use "Suite" as the label. **Unite** Use "Unite" as the label.
CanPreferHouseNum	In cases where the house number and postal code are both valid but in conflict, you can force the

optionName	Description/Valid Values
	postal code to be corrected based on the house number by specifying CanPreferHouseNum=Y. If you do not select this option the house number is changed to match the postal code. One of the following: **N** Change the house number to match the postal code. Default. **Y** Change the postal code to match the house number.
CanOutputCityAlias	Specifies whether or not to return the city alias when the alias is in the input address. This option is disabled when you specify CanOutputCityFormat=D. **Y** Output the city alias when the city alias is in the input. Default. **N** Never output the city alias even if it is in the input.
CanNonCivicFormat	Specifies whether or not non-civic keywords are abbreviated in the output. For example, Post Office Box vs. PO Box. **A** Abbreviate non-civic keywords. Default. **F** Do not abbreviate non-civic keywords. The full keyword is used.
EnableSERP	Specifies whether or not to use SERP options. **Y** Enable SERP options. **N** Do not enable SERP options. Default.
CanStandardAddressFormat	Specifies where to place secondary address information in the output address. Secondary address information refers to apartment numbers, suite numbers, and similar designators. **D** Place apartment information in the location specified in the StandardAddressFormat option. Default.

optionName	Description/Valid Values
	B Place apartment information at the at the end (back) of the AddressLine1 field. **F** Place the apartment number only (no label) at the beginning of the AddressLine1 field. For example, 400-123 Rue Main **E** Place the apartment number and label at the beginning of the AddressLine1 field. For example, Apt 400 123 Rue Main **S** Place apartment information on a separate line. **S** Place apartment information in the same location as the input address.
CanOutputCityFormat	Specifies whether to use the long, medium, or short version of the city if the city has a long name. For example, Long: BUFFALO HEAD PRAIRIE Medium: BUFFALO-HEAD-PR Short: BUFFALO-HD-PR **D** Use the default option specified by the OutputShortCityName option. Default. If you specify OutputShortCityName=V, the city is formatted as if you select `L` for this option (see below) and `Y` for **CanOutputCityAlias**. **S** Output short city name. **L** Output the long city name. **M** Output the medium city name. **I** Use the same city format as used in the input address. Output is L, M, or S.
CanRuralRouteFormat	Specifies where to place rural route delivery information. An example of an address with rural route delivery information is: 36 GRANT RD RR 3 ANTIGONISH NS In this address, "RR 3" is the rural route delivery information.

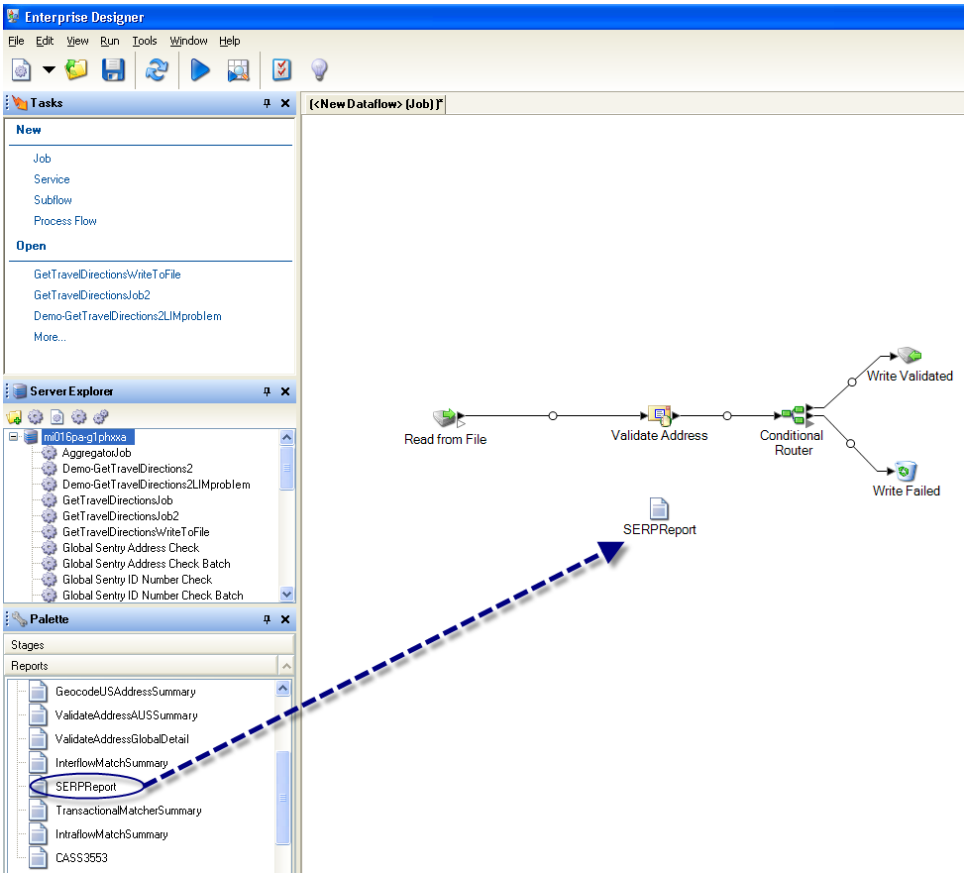
optionName	Description/Valid Values
	A Place rural route delivery information on the same line as the address, after the address information. Default. For example, 36 GRANT RD RR 3. **S** Place rural route delivery information on a separate address line. For example, 36 GRANT RD RR 3.
CanDeliveryOfficeFormat	Specifies where to place station information. An example of an address with station information is: PO BOX 8625 STN A ST. JOHN'S NL **I** Place station information in the same location as it is in the input address. Default. **A** Place station information on the same line as the address, after the address information. For example, PO BOX 8625 STN A **S** Place station information on a separate address line. For example, PO BOX 8625 STN A
CanDualAddressLogic	Specifies whether ValidateAddress should return a street match or a PO Box/non-civic match when the address contains both civic and non-civic information. One of the following: **D** Use DualAddressLogic Global Option. Default. Match to PO Box or other non-street data. **S** Match to street. For example, given the following input address: AddressLine1: 36 GRANT RD AddressLine2: RR 4 City: ANTIGONISH StateProvince: NS ValidateAddress would return one of the following: • If CanDualAddressLogic is set to S, ValidateAddress returns the following:

optionName	Description/Valid Values
	AddressLine1: 36 GRANT RD AddressLine2: RR 3 City: ANTIGONISH StateProvince: NS PostalCode: B2G 2L1 • If CanDualAddressLogic is set to P, ValidateAddress returns the following: AddressLine1: RR 4 City: ANTIGONISH StateProvince: NS PostalCode: B2G 2L2 The address data that is not used to standardize the address is returned in the **AdditionalInputData** field. For more information on this option, see **Output Data Options** on page 770.

SERP Processing

1. Validate Address must be in SERP Certified™ mode. If **(Not SERP Certified)** appears at the top of the window, click the **Enable SERP settings** button. The **Configure SERP** box will appear.
2. Click **Configure SERP**. The **SERP Report Fields** dialog box appears.
3. Type your merchant **CPC number**.
4. Type the mailer **Name, Address**, and **City, State, ZIP**.
5. Click **OK**.

6. In Enterprise Designer, drag the SERPReport from the Reports pallet to the canvas.



7. Double-click the **SERPReport** icon on the canvas.
8. On the **Stages** tab, ensure that the **Validate Address** checkbox is checked. Note that if you have renamed the Validate Address stage to something else, you should check the box with the name you have given the address validation stage.
9. On the **Parameters** tab, select the format for the report. You can create the report in PDF, HTML, or plain text format. PDF format is the default.
10. Click **OK**.

Obtaining SERP Return Codes

SERP return codes indicate the quality of the input address as determined by the Canada Post's Software Evaluation and Recognition Program regulations.

Option

To obtain SERP return codes, specify **OutputRecordTyp**e=P. For more information on OutputRecordType, see **Output Data Options** on page 770.

Output

SERP return codes are provided in the following output field.

Table 253: SERP Return Code Output

columnName	Description/Valid Values
CanadianSERPCode	Validation/correction return code (Canadian addresses only): **V** The input was valid. Canada Post defines a "valid" address as an address that meets all the following requirements: **Note:** There are exceptions. For further information, contact the CPC. • The address must contain all required components as found in CPC's Postal Code Data Files. • The address must provide an exact match on all components for only one address in CPC's Postal Code Data Files, allowing for acceptable alternate words and names listed in the CPC Postal Code Data Files. • Address components must be in a form that allows recognition without ambiguity. Certain components may require "qualifiers" to identify them. For instance, a Route Service address requires the key words "Rural Route" or "RR" for differentiation from a "Suburban Service" or "SS" address with the same number. **I** The input was invalid. An "invalid" address is one that does not meet CPC requirements for a valid address (see above). Examples of this include address components that are missing, invalid, or inconsistent. **C** The input was correctable. A "correctable" address is one that can be corrected to match one, and only one, address. **N** The input was non-correctable. A "non-correctable" address is one that could be corrected a number of different ways such that ValidateAddress cannot identify a single correct version. **F** The input address was foreign (outside of Canada).

International Address Options

Addresses outside of the U.S. and Canada are referred to as "international" addresses. The following options control international address processing:

optionName	Description/Valid Values
PerformInternationalProcessing	Specifies whether to process international addresses (addresses outside the U.S. and Canada). If you enable international address processing ValidateAddress will attempt to validate international addresses. If you disable international address processing, international addresses will fail, meaning they is returned with an "F" in the Status output field. The output field Status.Code will say "DisabledCoder." If you are not licensed for international address processing you must disable international address processing in order for your jobs to complete successfully, regardless of whether or not they contain international addresses. **Note:** You must have a valid license for international address processing to successfully process international addresses. If you enable international address processing but are not licensed for this feature, or your license has expired, you will receive an error. One of the following: **N** No, do not process international addresses. **Y** Yes, process international addresses (default).
Database.International	Specifies which database you want to use for validating international addresses. To specify a database for international address validation, select a database in the **Database** drop-down list. Only databases that have been defined in the **INTL Database Resources** panel in the Management Console are available.
InternationalCityStreetSearching	By default, ValidateAddress provides a balance of good address matching accuracy with good performance. If you are willing to trade matching accuracy for faster performance, use the InternationalCityStreetSearching option to increase processing speed. When you do this, some

optionName	Description/Valid Values
	accuracy is lost. This option only controls performance for addresses outside the U.S. and Canada. This setting affects a small percentage of records, mostly addresses in the U.K. There is no performance control for U.S. and Canadian address processing. If you use GetCandidateAddresses, the candidate addresses returned by GetCandidateAddresses may differ from the multiple matches returned by ValidateAddress if you set the performance tuning option for international addresses to any value other than 100. To control performance, specify a value from 0 to 100. A setting of 100 maximizes accuracy while a setting of 0 maximizes speed. The default is 100.
AddressLineSearchOnFail	This option enables ValidateAddress to search the AddressLine input fields for the city, state/province, postal code, and country when the address cannot be matched using the values in the City, StateProvince, and PostalCode input fields. Consider enabling this option if your input addresses have the city, state/province, and postal code information in the AddressLine fields. Consider disabling this option if your input addresses use the City, State/Province and PostalCode fields. If you enable this option and these fields are used, there is an increased possibility that ValidateAddress will fail to correct values in these fields (for example a misspelled city name). One of the following: **N** No, do not search the AddressLine fields. **Y** Yes, search the address line fields. Default.

Output

The output from ValidateAddress contains different information depending on the output categories you select.

The output categories are:

- **Standard Address Output** on page 803
- **Parsed Address Elements Output** on page 805
- **Parsed Input** on page 807
- **Postal Data Output** on page 809
- **Result Indicators** on page 811
- **Output from Options** on page 825
- **About AdditionalInputData** on page 830

Note: The output is returned in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Standard Address Output

Standard address output consists of four lines of the address which correspond to how the address would appear on an actual address label. City, state/province, postal code, and other data is also included in standard address output. ValidateAddress returns standard address output for validated addresses if you set **OutputRecordTyp**e = **A**. Standard address fields are always returned for addresses that could not be validated regardless of whether or not you set **OutputRecordTyp**e = **A**. For non-validated addresses, the standard address output fields contain the address as it appeared in the input ("pass through" data). If you want ValidateAddress to standardize address according to postal authority standards when validation fails, specify **OutputFormattedOnFail** = Y in your request.

Table 254: Standard Address Output

columnName	Description / Valid Values
AdditionalInputData	Input data not used by ValidateAddress. For more information, see **About AdditionalInputData** on page 830.
AddressLine1	If the address was validated, the first line of the validated and standardized address. If the address could not be validated, the first line of the input address without any changes.
AddressLine2	If the address was validated, the second line of the validated and standardized address. If the address could not be validated, the second line of the input address without any changes.
AddressLine3	If the address was validated, the third line of the validated and standardized address. If the address could not be validated, the third line of the input address without any changes.

columnName	Description / Valid Values
AddressLine4	If the address was validated, the fourth line of the validated and standardized address. If the address could not be validated, the fourth line of the input address without any changes.
AddressLine5	For U.K. addresses only. If the address was validated, the fifth line of the validated and standardized address. If the address could not be validated, the fifth line of the input address without any changes.
City	The validated city name.
Country	The country in the format determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English • French • German • Spanish
DepartmentName	For U.K. addresses only, a subdivision of a firm. For example, Engineering Department.
FirmName	The validated firm or company name.
PostalCode	The validated ZIP Code™ or postal code.
PostalCode.AddOn	The 4-digit add-on part of the ZIP Code™. For example, in the ZIP Code™ 60655-1844, 1844 is the 4-digit add-on. (U.S. addresses only.)
PostalCode.Base	The 5-digit ZIP Code™; for example 20706 (U.S. addresses only).
StateProvince	The validated state or province abbreviation.
USUrbanName	The validated urbanization name. (U.S. addresses only.)

Parsed Address Elements Output

Output addresses are formatted in the parsed address format if you set **OutputRecordType** = **E**. If you want ValidateAddress to return formatted data in the Parsed Address format when validation fails (that is, a normalized address), specify **OutputFormattedOnFail** = Y.

Note: If you want ValidateAddress to always return parsed input data regardless of whether or not validation is successful, specify **OutputRecordType = I**. For more information, see **Parsed Input** on page 807.

Table 255: Parsed Address Output

columnName	Description / Valid Values
AdditionalInputData	Input data not used by ValidateAddress. For more information, see **About AdditionalInputData** on page 830.
ApartmentLabel	Apartment designator (such as STE or APT), for example: 123 E Main St **APT** 3
ApartmentLabel2	Secondary apartment designator, for example: 123 E Main St APT 3, 4th **Floor** **Note:** In this release, this field will always be blank.
ApartmentNumber	Apartment number. For example: 123 E Main St APT **3**
ApartmentNumber2	Secondary apartment number. For example: 123 E Main St APT 3, **4th** Floor **Note:** In this release, this field will always be blank.
CanadianDeliveryInstallationAreaName	Delivery installation name (Canadian addresses only)
CanadianDeliveryInstallationQualifierName	Delivery installation qualifier (Canadian addresses only)
CanadianDeliveryInstallationType	Delivery installation type (Canadian addresses only)
City	Validated city name

columnName	Description / Valid Values
Country	Country. Format is determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English • French • German • Spanish
DepartmentName	For U.K. addresses only, a subdivision of a firm. For example, Engineering Department.
FirmName	The validated firm or company name
HouseNumber	House number, for example: **123** E Main St Apt 3
LeadingDirectional	Leading directional, for example: 123 **E** Main St Apt 3
POBox	Post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode	Validated postal code. For U.S. addresses, this is the ZIP Code.
PrivateMailbox	Private mailbox indicator.
PrivateMailbox.Type	The type of private mailbox. Possible values include: • Standard • Non-Standard **Note:** This replaces PrivateMailboxType (no period in field name). Please modify your API calls accordingly.
RRHC	Rural Route/Highway Contract indicator
StateProvince	Validated state or province name

columnName	Description / Valid Values
StreetName	Street name, for example: 123 E **Main** St Apt 3
StreetSuffix	Street suffix, for example: 123 E Main **St** Apt 3
TrailingDirectional	Trailing directional, for example: 123 Pennsylvania Ave **NW**
USUrbanName	USPS® urbanization name. Puerto Rican addresses only.

Parsed Input

The output can include the input address in parsed form. This type of output is referred to as "parsed input." Parsed input fields contain the address data that was used as input regardless of whether or not ValidateAddress validated the address. Parsed input is different from the "parsed address elements" output in that parsed address elements contain the validated address if the address could be validated, and, optionally, the input address if the address could not be validated. Parsed input always contains the input address regardless of whether or not ValidateAddress validated the address.

To include parsed input fields in the output, set **OutputRecordType** = **I**.

Table 256: Parsed Input

columnName	Description / Valid Values
ApartmentLabel.Input	Apartment designator (such as STE or APT), for example: 123 E Main St **APT** 3
ApartmentNumber.Input	Apartment number, for example: 123 E Main St APT **3**
CanadianDeliveryInstallationAreaName.Input	Delivery installation name (Canadian addresses only)
CanadianDeliveryInstallationQualifierName.Input	Delivery installation qualifier (Canadian addresses only)
CanadianDeliveryInstallationType.Input	Delivery installation type (Canadian addresses only)
City.Input	Validated city name

columnName	Description / Valid Values
Country.Input	Country. Format is determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English • French • German • Spanish
FirmName.Input	The validated firm or company name
HouseNumber.Input	House number, for example: **123** E Main St Apt 3
LeadingDirectional.Input	Leading directional, for example: 123 **E** Main St Apt 3
POBox.Input	Post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode.Input	Validated postal code. For U.S. addresses, this is the ZIP Code.
PrivateMailbox.Input	Private mailbox indicator
PrivateMailbox.Type.Input	The type of private mailbox. Possible values include: • Standard • Non-Standard
RRHC.Input	Rural Route/Highway Contract indicator
StateProvince.Input	Validated state or province name
StreetName.Input	Street name, for example: 123 E **Main St** Apt 3
StreetSuffix.Input	Street suffix, for example: 123 E Main St Apt 3
TrailingDirectional.Input	Trailing directional, for example: 123 Pennsylvania Ave **NW**

columnName	Description / Valid Values
USUrbanName.Input	USPS® urbanization name

Postal Data Output

If**OutputRecordType** contains **P** then the following columns are returned in the output.

Table 257: Postal Data Output

columnName	Description / Valid Values
CanadianSERPCode	Validation/correction return code (Canadian addresses only). For more information, see **Obtaining SERP Return Codes** on page 799.
IntHexaviaCode	For addresses in France only, a numeric code that represents the street. For information about Hexavia codes, see **www.laposte.fr**.
IntINSEECode	For addresses in France only, a numeric code that represents the city. For a listing of INSEE codes, see **www.insee.fr**.
PostalBarCode	The two-digit delivery point portion of the delivery point barcode (U.S. addresses only) For more information, see **Creating Delivery Point Barcodes** on page 775.
USAltAddr	Indicates whether or not alternate address matching logic was used, and if so which logic was used (U.S. addresses only). One of the following: **null** No alternate address scheme used. **D** Delivery point alternate logic was used. **E** Enhanced highrise alternate match logic was used. **S** Small town default logic was used. **U** Unique ZIP Code logic was used.
USBCCheckDigit	Check-digit portion of the 11-digit delivery point barcode (U.S. addresses only). For more information, see **Creating Delivery Point Barcodes** on page 775.
USCarrierRouteCode	Carrier route code (U.S. addresses only). For more information, see **Obtaining Carrier Route Codes** on page 775.

columnName	Description / Valid Values
USCongressionalDistrict	Congressional district (U.S. addresses only). For more information, see **Obtaining Congressional Districts** on page 773.
USCountyName	County name (U.S. addresses only). For more information, see **Obtaining County Names** on page 774.
USFinanceNumber	The finance number in which the address resides (U.S. addresses only). The finance number is a number assigned by the USPS to an area that covers multiple ZIP Codes. ValidateAddress will successfully validate an address only if its finance number matches the finance number of the candidate address in the U.S. Database.
USFIPSCountyNumber	FIPS (Federal Information Processing Standards) county number (U.S. addresses only). For more information, see **Obtaining FIPS County Numbers** on page 774.
USLACS	Indicates whether or not the address is a candidate for LACS^{Link} conversion (U.S. addresses only). One of the following: **Y** Yes, the address is a candidate for LACS^{Link} processing. If LACS^{Link} is enabled, ValidateAddress will attempt to convert the address using the LACS^{Link} database. If the conversion attempt is successful, the output address is the new address obtained from the LACS^{Link} database. If the attempt is not successful, the address will not be converted. **N** No, the address is not a candidate for LACS^{Link} processing. LACS^{Link} processing may still be attempted if LACS^{Link} processing is requested, the LACS^{Link} database is installed, and one of the following is true: • The address matches to a Rural Route address and the RecordType.Default field returns a Y. • The input address could not be matched to any address in the U.S. Postal Database (Failures due to multiple matches are not LACS^{Link} candidates.)
USLastLineNumber	A six-character alphanumeric value that groups together ZIP Codes that share the same primary city. For example, addresses with the following two last lines would have the same last line number: Chantilly VA 20151 Chantilly VA 20152

Result Indicators

Result indicators provide information about the kinds of processing performed on an address. There are two types of result indicators:

- **Record-Level Result Indicators** on page 811
- **Field-Level Result Indicators** on page 815

Record-Level Result Indicators

Record-level result indicators provide data about the results of ValidateAddress processing for each record, such as the success or failure of the match attempt, which coder processed the address, and other details. The following table lists the record-level result indicators returned by ValidateAddress.

Table 258: Record Level Indicators

columnName	Description / Valid Values
AddressFormat	The type of address data being returned: **F** French format (for example: 123 Rue Main) **E** English format (for example: 123 Main St)
Confidence	The level of confidence assigned to the address being returned. Range is from zero (0) to 100; zero indicates failure, 100 indicates a very high level of confidence that the match results are correct. For multiple matches, the confidence level is 0. For details about how this number is calculated, see **Introduction to the Validate Address Confidence Algorithm** on page 889.
CouldNotValidate	If no match was found, which address component could not be validated: • ApartmentNumber • HouseNumber • StreetName • PostalCode • City • Directional • StreetSuffix • Firm • POBoxNumber • RuralRoute

columnName	Description / Valid Values
	Note: More than one component may be returned, in a comma-separated list.
CountryLevel	The category of address matching available. This is always "A" for U.S. and Canadian addresses. One of the following: **A** The address is in a country for which there is highly detailed postal data available. Addresses in this match level can have the following address elements validated and corrected, and added if missing from the input: • Postal code • City name • State/county name • Street address elements • Country name. **B** The address is in a country for which there is a medium level of postal data available. Addresses in this match level can have the following address elements validated and corrected, and added if missing from the input: • Postal code • City name • State/county name • Country name **C** The address is in a country for which the postal data is least detailed. Addresses in this match level can have the following actions performed on them: • Validate and correct country name (cannot supply missing country name) • Validate the format of the postal code (cannot supply missing postal code or validate the code)
MatchScore	MatchScore provides an indication of the degree to which the output address is correct. It is significantly different from Confidence in that Confidence is indicates how much the input address changed to obtain a match, whereas the meaning of Match Score varies between U.S. and non-U.S. addresses.

columnName	Description / Valid Values
	For U.S. addresses, MatchScore is a one-digit score on a scale of 0 to 9 that reflects the closeness of the street-name match (after transformations by ValidateAddress, if any). Zero indicates an exact match and 9 indicates the least likely match. If no match was found, this field is blank. For non-U.S. and non-Canadian addresses, MatchScore is a five-digit score, with a maximum value of 00999. Higher numbers indicates a closer match. This field does not apply to Canadian addresses. Note that you cannot equate match scores from U.S. addresses with those of non-U.S. addresses. For example, a match score of 4 for a U.S address does not indicate the same level of match as a 00004 for a non-U.S. address. **Note:** The Validate Address and Advanced Matching Module components both use the MatchScore field. The MatchScore field value in the output of a dataflow is determined by the last stage to modify the value before it is sent to an output stage. If you have a dataflow that contains Validate Address and Advanced Matching Module components and you want to see the MatchScore field output for each stage, use a Transformer stage to copy the MatchScore value to another field. For example, Validate Address produces an output field called MatchScore and then a Transformer stage copies the MatchScore field from Validate Address to a field called AddressMatchScore. When the matcher stage runs it populates the MatchScore field with the value from the matcher and passes through the AddressMatchScore value from Validate Address.
MultimatchCount	If multiple matches were found, indicates the number of records that are possible matches.
MultipleMatches	Indicates which address component had multiple matches, if multiple matches were found: • Firm • LeadingDirectional • PostalCode • StreetName • StreetSuffix

columnName	Description / Valid Values
	• TrailingDirectional • Urbanization **Note:** More than one component may be returned, in a comma-separated list.
ProcessedBy	Which address coder processed the address: **USA** U.S. address coder **CAN** Canadian address coder **INT** International address coder
RecordType	Type of address record, as defined by U.S. and Canadian postal authorities (supported for U.S. and Canadian addresses only): • FirmRecord • GeneralDelivery • HighRise • PostOfficeBox • RRHighwayContract • Normal
RecordType.Default	Code indicating the "default" match: **Y** The address matches a default record. **null** The address does not match a default record.
Status	Reports the success or failure of the match attempt. For multiple matches, this field is "F" for all the possible matches. **null** Success **F** Failure
Status.Code	Reason for failure, if there is one. For multiple matches, all possible matches is "MultipleMatchesFound." • DisabledCoder • InsufficientInputData • MultipleMatchesFound • UnableToValidate

columnName	Description / Valid Values
Status.Description	Description of the problem, if there is one. **Possible Multiple Addresses Found** — This value will appear if Status.Code=MultipleMatchesFound. **Address Not Found** — This value will appear if Status.Code=UnableToValidate. **PerformUSProcessing disabled** — This value will appear if Status.Code=DisabledCoder. **PerformCanadianProcessing disabled** — This value will appear if Status.Code=DisabledCoder. **PerformInternationalProcessing disabled** — This value will appear if Status.Code=DisabledCoder.

Field-Level Result Indicators

Field-level result indicators describe how ValidateAddress handled each address element. Field-level result indicators are returned in the qualifier "Result". For example, the field-level result indicator for HouseNumber is contained in **HouseNumber.Result**.

To enable field-level result indicators, specify **OutputFieldLevelReturnCodes=Y**. For more information on this option, see **Output Data Options** on page 770.

The following table lists the field-level result indicators. If a particular field does not apply to an address, the result indicator may be blank.

Table 259: Field-Level Result Indicators

columnName	Valid Values
AddressRecord.Result	These result codes apply to international addresses only. **M** — Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. **S** — Standardized. This option includes any standard abbreviations. **U** — Unmatched. **V** — Validated. The data was confirmed correct and remained unchanged from input.

columnName	Valid Values	
ApartmentLabel.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only.
	R	The apartment label is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
ApartmentNumber.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. addresses that are an EWS match will have a value of P. U.S. and Canadian addresses only.
	R	The apartment number is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.

columnName	Valid Values	
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
City.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	F	Hyphens missing or punctuation errors. Canadian addresses only.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output.
	R	The city is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
Country.Result	These result codes do not apply to U.S. or Canadian addresses.	
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
FirmName.Result	**C**	Corrected. U.S. addresses only.

columnName	Valid Values	
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only.
	U	Unmatched. U.S. and Canadian addresses only.
	V	Validated. The data was confirmed correct and remained unchanged from input. U.S. addresses only.
HouseNumber.Result	**A**	Appended. The field was added to a blank input field. Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	O	Out of range. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	R	The house number is required but is missing from the input address. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. or Canadian addresses.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
LeadingDirectional.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. Non-blank input was corrected to a non-blank value. U.S. addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.

columnName	Valid Values	
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input. Does not apply to Canadian addresses.
POBox.Result	**A**	Appended. The field was added to a blank input field. Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple matches. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	R	The P.O. Box number is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
PostalCode.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.

columnName	Valid Values	
	C	Corrected. U.S. and Canadian addresses only.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Does not apply to U.S. addresses.
	R	The postal code is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. or Canadian addresses.
	U	Unmatched. For example, if the street name does not match the postal code, both StreetName.Result and PostalCode.Result will contain U.
	V	Validated. The data was confirmed correct and remained unchanged from input.
PostalCodeCity.Result	These result codes apply to international addresses only.	
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
PostalCode.Source	These result codes apply to U.S. addresses only.	
	FinanceNumber	The ZIP Code™ in the input was verified by using USPS® Finance Number groupings.

columnName	Valid Values	
	ZIPMOVE	The ZIP Code™ in the input address was corrected because the USPS® redrew ZIP Code™ boundaries and the address is now in a different ZIP Code™.
PostalCode.Type	**P**	The ZIP Code™ contains only PO Box addresses. U.S. addresses only.
	U	The ZIP Code™ is a unique ZIP Code™ assigned to a specific company or location. U.S. addresses only.
	M	The ZIP Code™ is for military addresses. U.S. addresses only.
	null	The ZIP Code™ is a standard ZIP Code™.
RRHC.Result	**C**	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.
	M	Multiple matches. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	R	The rural route/highway contract is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. U.S. and Canadian addresses only.
	U	Unmatched. U.S. and Canadian addresses only.
	V	Validated. The data was confirmed correct and remained unchanged from input. U.S. and Canadian addresses only.
RRHC.Type	These result codes apply to U.S. addresses only.	
	HC	The address is a Highway Contract address.
	RR	The address is a Rural Route address.
StateProvince.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.

columnName	Valid Values	
	C	Corrected. U.S. addresses only.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only.
	R	The state is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
Street.Result	These result codes apply to international addresses only.	
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output.
	R	Street corrected. House number is out of range. Applies to UK and Japanese records only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
StreetName.AbbreviatedAlias.Result	Indicates the result of abbreviated alias processing. One of the following:	
	null	No abbreviated alias processing attempted.
	B	The StreetName field contains the base street name.
	L	The standardized address length is less than 31 characters so the StreetName field contains the base name.

columnName	Valid Values
	N No abbreviated alias found. **Y** An abbreviated alias was found for input address. The StreetName field contains the abbreviated alias.
StreetName.Alias.Type	This result code applies to U.S. addresses only. **Note:** In previous releases this field was named StreetName.AliasType with no "." between "Alias" and "Type." This old name is obsolete. Please update your processes to use the new name StreetName.Alias.Type. **Abbreviated** The alias is an abbreviation of the street name. For example, HARTS-NM RD is an abbreviated alias for HARTSVILLE NEW MARLBORO RD. **Changed** There has been an official street name change and the alias reflects the new name. For example if SHINGLE BROOK RD is changed to CANNING DR, then CANNING DR would be a changed alias type. **Other** The street alias is made up of other names for the street or common abbreviations of the street. **Preferred** The street alias is the locally preferred alias. For example, a street is named "South Shore Dr." because it runs along the southern shore of a lake, not because it is south of a municipal demarcation line. So, "South" is not a predirectional in this case and should not be shorted to "S". So, "South Shore Dr." would be the preferred alias.
StreetName.PreferredAlias.Result	Indicates the result of preferred alias processing. One of the following: **null** No preferred alias processing attempted. **A** Preferred alias processing was not attempted because the input address matched to an alias. Preferred alias processing is only attempted for base addresses. **N** No preferred alias found. **Y** A preferred alias was found for the input address. The StreetName field contains the preferred alias.
StreetName.Result	**A** Appended. The field was added to a blank input field. Canadian addresses only.

columnName	Valid Values	
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Does not apply to U.S. addresses.
	S	Standardized. This option includes any standard abbreviations. U.S. and Canadian addresses only.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
StreetSuffix.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched. Does not apply to U.S. addresses.

columnName	Valid Values	
	V	Validated. The data was confirmed correct and remained unchanged from input.
TrailingDirectional.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About AdditionalInputData** on page 830.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
USUrbanName.Result	These result codes apply to U.S. addresses only.	
	A	Appended. The field was added to a blank input field.
	C	Corrected.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.

Output from Options

ValidateAddress returns additional data depending on the options you select. For information on the output generated by each option, see the options listed in the following sections:

Enhanced Line of Travel Output

Enhanced Line of Travel processing produces the following output.

columnName	Description/Valid Values
USLOTCode	Line of Travel sequence code and an indicator denoting USPS® LOT sequence. This field is in the format nnnnY where: **nnnn** The four-digit LOT code. **Y** One of the following: • **A**—Ascending LOT sequence • **D**—Descending LOT sequence
USLOTHex	A hexadecimal value that allows you to sort your file in ascending order only. The hexadecimal values range from 0 to FF ascending, then FF through 0 descending.
USLOTSequence	A two-byte value used for final sortation in place of the DPC add-on. It consists of an uppercase letter followed by a digit 0 through 9. Values range from A0 (99 descending) through J9 (00 descending), and K0 (00 ascending) through T9 (99 ascending).

LACS^{Link} Output

columnName	Description/Valid Values
USLACS	Indicates whether or not the address is a candidate for LACS^{Link} conversion (U.S. addresses only). One of the following: **Y** Yes, the address is a candidate for LACS^{Link} processing. If LACS^{Link} is enabled, ValidateAddress will attempt to convert the address using the LACS^{Link} database. If the conversion attempt is successful, the output address is the new address obtained from the LACS^{Link} database. If the attempt is not successful, the address will not be converted. **N** No, the address is not a candidate for LACS^{Link} processing. LACS^{Link} processing may still be attempted if LACS^{Link} processing is requested, the LACS^{Link} database is installed, and one of the following is true: • The address matches to a Rural Route address and the RecordType.Default field returns a Y.

columnName	Description/Valid Values
	• The input address could not be matched to any address in the U.S. Postal Database (Failures due to multiple matches are not LACS^{Link} candidates.)
USLACS.ReturnCode	Indicates the success or failure of LACS^{Link} processing. (U.S. addresses only.) **A** LACS^{Link} processing successful. Record matched through LACS^{Link} processing. **00** LACS^{Link} processing failed. No matching record found during LACS^{Link} processing. **09** LACS^{Link} processing matched the input address to an older highrise default address. The address has been converted. Rather than provide an imprecise address, LACS^{Link} processing does not provide a new address. **14** LACS^{Link} processing failed. Match found during LACS^{Link} processing but conversion did not occur due to other USPS® regulations. **92** LACS^{Link} processing successful. Record matched through LACS^{Link} processing. Unit number dropped on input. **null** LACS^{Link} did not process the record, or LACS^{Link} processing was not attempted.

RDI Output

columnName	Description/Valid Values
RDI	Return values indicating address type. **B** The address is a business address. **R** The address is a residential address. **M** The address is both a residential and a business address. **null** Not checked because the address did not code at a ZIP + 4® level, or RDI™ was not performed.

DPV and CMRA Output

columnName	Description/Valid Values
DPV	Indicates the results of Delivery Point Validation (DPV) processing. **Y** DPV confirmed. **N** Address is not deliverable. **S** The primary number was validated but the secondary number could not be confirmed. **D** The primary number was validated but the secondary number was missing from input. **M** The address matches multiple valid delivery points. **U** The address could not be confirmed because the address did not code at the ZIP + 4® level. **V** The address caused a false-positive violation. For more information, see **Reporting DPV False-Positive Violations** on page 886.
CMRA	Indicates if the address is a Commercial Mail Receiving Agency (CMRA) **Y** Yes, the address is a CMRA. **N** No, the address is not a CMRA. **U** Unconfirmed.
DPVFootnote	DPV footnote codes. **AA** Input address matched to the ZIP + 4® file. **A1** Input address not matched to the ZIP + 4® file. **BB** Input address matched to DPV (all components). **CC** Input address primary number matched to DPV but secondary number not match (present but not valid). **N1** Input address primary number matched to DPV but high rise address missing secondary number. **M1** Input address primary number missing. **M3** Input address primary number invalid. **P1** Input address missing PO, RR or HC Box number. **RR** Input address matched to CMRA.

columnName	Description/Valid Values
	R1 Input address matched to CMRA but secondary number not present.
DPVVacant	Indicates whether the building is vacant (unoccupied for 90 days). One of the following: **Y** Yes, the building is vacant. **N** No, the building is not vacant. **null** The **DPVDetermineVacancy** option was not turned on.
DPVNoStat	Indicates whether the building is a "no stat" building and therefore unable to receive mail. One of the following: **Y** Yes, the building is a "no stat" building, which means the building is not receiving mail. **N** No, the building is not a "no stat" building, which means the building does receive mail. **null** The **DPVDetermineNoStat** option was not turned on.

Suite^{Link} Output

columnName	Description/Valid Values
SuiteLinkReturnCode	Indicates whether or not ValidateAddress corrected the secondary address information (U.S. addresses only). One of the following: **A** ValidateAddress corrected the secondary address information. **00** ValidateAddress did not correct the secondary address information. **null** Suite^{Link} was not performed. **XX** Suite^{Link} processing encountered an error. For example, an error would occur if the Suite^{Link} database is expired.
SuiteLinkMatchCode	Provides additional information on the Suite^{Link} match attempt. (U.S. addresses only) **A** ValidateAddress corrected the secondary address information.

columnName	Description/Valid Values
	B ValidateAddress did not correct the secondary address information. No additional detail about the match attempt is available. **C** The words in the FirmName field are all "noise" words. Noise words are defined by the USPS® and are ignored when attempting to mach the firm name. Examples of noise words are "company" and "corporation". ValidateAddress is not able to correct secondary address information for firm names that consist entirely of noise words. For example "Company and Corporation" is all noise words. **D** The address is not a high-rise default address. SuiteLink matching is only done for high-rise default addresses. A high-rise default is a default to use when the address does not contain valid secondary information (the apartment number or apartment type is missing). **E** SuiteLink processing failed because the SuiteLink database is expired. **null** SuiteLink was not performed or there was an error.
SuiteLinkFidelity	Indicates how well ValidateAddress matched the firm name to the firm names in the SuiteLink database. **1** The firm name matches the SuiteLink database exactly. **2** Good match. All words in the firm name except one matched the firm name in the SuiteLink database. **3** Poor match. More than one word in the firm name did not match the firm name in the SuiteLink database. **null** SuiteLink could not match the firm name, or was not performed, or there was an error.

About AdditionalInputData

ValidateAddress ignores some input data during the address standardization process. This extraneous data (sometimes referred to as "dropped data") is returned in the AdditionalInputData column. Some examples of dropped data include:

- Delivery instructions (for example, "Leave at back door")
- Phone numbers (for example, "555-135-8792")
- Attention lines (for example, "Attn: John Smith")

Data such as this is generally not embedded in an address. If it is embedded, ValidateAddress can usually identify this extraneous data and return it in the AdditionalInputData column.

Note: ValidateAddress does not return dropped data from split indicia addresses. A split indicia address is one where a primary address is split between multiple address lines. For example, if the primary address is "1 Green River Valley Rd" then the following would be a split indicia version of this address: 1 Green River Valley Rd 01230

If there is more than one piece of dropped data in an address, each piece of data is separated by a semicolon and a space ("; ") for U.S. addresses and a space for addresses outside the U.S. The order of dropped data in AdditionalInputData is:

1. Care of, mail stop (U.S. addresses only)
2. Other extraneous data found on address lines
3. Entire unused data lines

For example, if this is the input address:

123 Main St C/O John Smith
Apt 5 Drop at back dock
jsmith@something.com
555-123-4567
05674

Then AdditionalInputData would contain:

C/O John Smith; Apt 5 Drop At Back Dock; 555-123-4567; Jsmith@g1.Com; 555-123-4567

ValidateAddress can handle the following types of extraneous data:

- **Care Of Data** on page 831
- **Extraneous Data on Its Own Address Line** on page 831
- **Extraneous Data Within an Address Line** on page 832
- **Dual Addresses** on page 833

Care Of Data

For U.S. addresses only, "care of" data is returned in AdditionalInputData. The following addresses contain examples of "care of" data:

123 Main St C/O John Smith
Apt 5
05674

123 Main St
Apt 5 ATTN John Smith
05674

123 Main St Apt 5
MailStop 2
05674

Extraneous Data on Its Own Address Line

ValidateAddress returns extraneous data on its own address line for U.S. and Canadian addresses.

For U.S. addresses, ValidateAddress uses the first two non-blank address lines to perform address standardization, unless either the firm name extraction or urbanization code extraction options are enabled (see **Address Line Processing for U.S. Addresses** on page 769 for more information). Data on other address lines is returned in AdditionalInputData. In the following address, "John Smith" would be returned in AdditionalInputData because it is in the third non-blank address line and ValidateAddress only uses the first two non-blank address lines for U.S. addresses.

123 Main St
Apt 5
John Smith
05674

If one of either of the first two non-blank address lines contains extraneous data, that data is returned in AdditionalInputData. For example, in the following addresses "John Smith" would be returned in AdditionalAddressData.

123 Main St
John Smith
05674

John Smith
123 Main St
05674

In the following address both "John Smith" and "Apt 5" would both be returned in AdditionalInputData. "John Smith" would be returned because it is extraneous data in one of the first two address lines and "Apt 5" would be returned because U.S. address data must be in the first two non-blank address lines.

John Smith
123 Main St
Apt 5
05674

Extraneous Data Within an Address Line

Extraneous data that is within an address line is returned in AdditionalInputData. For example, in the following addresses "John Smith" would be returned in AdditionalInputData.

123 Main St John Smith
05674

123 Main St Apt 5 John Smith
05674

123 Main St John Smith
Apt 5
05674

123 Main St
Apt 5 John Smith
05674

For U.S. addresses, only extraneous data at the end of the address line is returned in AdditionalInputData. Extraneous data that is not at the end of an address line is not returned for U.S. addresses. For example, in the following addresses "John Smith" is not returned.

John Smith 123 Main St
05674

123 Main John Smith St
05674

The AdditionalInputData column will sometimes contain the original street name or suffix if the street name was changed to obtain a match and the street name or suffix was at the end of a line. For example this address:

Pitney Bowes Software
4200 Parlament
Lanham MD

ValidateAddress would correct the spelling of the street name and add the suffix, returning "4200 Parliament Pl" as the corrected street address and "Parlament" in AdditionalInputData.

Dual Addresses

A dual address is an address that contains both street and PO Box/Rural Route/Highway Contract information. Depending on the processing options you select, the portion of the dual address that is not used for address standardization may be returned in AdditionalInputData. For more information, see **About Dual Address Logic** on page 781.

Reports

USPS CASS 3553 Report

The USPS CASS 3553 report must be given to the USPS along with the mailing to qualify for certain discounts. The report contains information about the software you used for CASS processing, information about your name-and-address list, information about your output file, information about the mailer, and other statistics about your mailing. For detailed information about USPS Form 3553, see **www.usps.com**.

For more information about CASS settings see **CASS Certified Processing** on page 791. For instructions on how to use reports, see the *Spectrum™ Technology Platform User's Guide*.

Validate Address Summary Report

The Validate Address Summary Report lists statistics about the job, such as the total number of records processed, the number of addresses validated, and more. For instructions on how to use reports, see the *Spectrum™ Technology Platform User's Guide*.

ValidateAddressAUS

ValidateAddressAUS standardizes and validates Australian addresses using Australia Post address data. It also adds missing postal information, such as postal codes, city names, state/territory names, and more.

ValidateAddressAUS also returns result indicators about validation attempts, such as whether or not ValidateAddressAUS validated the address, and the reason for failure if the address could not be validated.

During address matching and standardization, ValidateAddressAUS separates address lines into components and compares them to the contents of a Universal Addressing Module database. If a match is found, the input address is *standardized* to the database information.

ValidateAddressAUS is part of the Universal Addressing Module. For more information on this module, including a listing of other components included with it, see **What Is the Universal Addressing Module?** on page 722.

Input

ValidateAddressAUS takes a standard address as input. All addresses use this format.

Table 260: Input Format

columnName	Format	Description
AddressLine1	String [288]	The first address line.
AddressLine2	String [288]	The second address line.
AddressLine3	String [288]	The third address line.
AddressLine4	String [288]	The fourth address line.
City	String [48]	The city/locality/suburb name. This can optionally be entered into one of the AddressLine fields along with the State and Postal Code.
StateProvince	String [4]	The state. This can optionally be entered into one of the AddressLine fields along with the City and Postal Code.
PostalCode	String [8]	The postal code. This can optionally be entered into one of the AddressLine fields along with the State and City.

Options

ValidateAddressAUS provides several options that enable you to control how addresses are processed and the type of information returned.

Table 261: Options

optionName	Description/Valid Values
Database	Specifies the database to be used for Australian address validation. Only databases that have been defined in the **Australia Database Resources** panel in the Management Console are available.
OutputFieldLevelReturnCodes	Outputs result fields associated with certain output elements. See **Result Codes** on page 837. Valid values are: **N** No, do not include result codes for individual fields in the output (default). **Y** Yes, include result codes for individual fields in the output.
OutputOriginalInputFields	Returns the original input data. See **Original Input Data** on page 839. Valid values are: **N** No, do not include original input data in the output (default). **Y** Yes, include original input data in the output.
OutputMatchedAddressFields	Returns parsed address elements. See **Parsed Address Elements** on page 838. Valid values are: **N** No, do not include parsed address elements in the output (default). **Y** Yes, include parsed address elements in the output.
AmasFormatting	Specifies that output address data is to be formatted using Address Matching Approval System (AMAS) conventions. This option causes Validate Address AUS to use AMAS rules when standardizing an address. AMAS is an Australia Post program for enforcing addressing standards. For more information on the AMAS formatting conventions, refer to the Address Matching Approval System (AMAS) Handbook.

optionName	Description/Valid Values
	This option modifies the output data as follows. • Numeric fields are padded with zeros. This affects the following output fields: HouseNumber, HouseNumber2, PostalDeliveryNumber, and DPID. For example, if the input address is 298 New South Head Rd Double Bay NSW 2028, then the format of the HouseNumber field is changed from 298 to 00298. • If a match is not made, then all digits in the DPID field will be zero. For example, 00000000. • If a match is not made, then all return fields (parsed address elements) will be blank, except numeric fields which will contain all zeros. • The CCD field is not output. Valid values are: **N** No, do not format the output data using AMAS conventions (default). **Y** Yes, format the output data using AMAS conventions.

Output

At a minimum, the output of ValidateAddressAUS consists of the standard output fields listed in **Standard Output Fields** on page 836. In addition to these standard fields, the output may also include other information, depending on the output options you select. For more information on the optional output fields, see **Result Codes** on page 837, **Parsed Address Elements** on page 838, and **Original Input Data** on page 839.

Standard Output Fields

The following table lists the standard fields that are output by ValidateAddressAUS.

Table 262: Output Fields

columnName	Description
AddressLine1	A formatted address line.
BuildingName	The building name.
City	The city/locality/suburb name 1.
City2	The city/locality/suburb name 2 - split names e.g. VIA.

columnName	Description
StateProvince	The state.
PostalCode	The postal code.
CCD	The Census Collection District. The basic geographic unit for collection, processing and output of census data. In general, there are about 200 to 250 households per CCD, and about 37,000 CCDs throughout Australia.
DPID	The Delivery Point Identifier. An eight digit number from the Australia Post Postal Address File that uniquely identifies a mail delivery point, such as a street address.
Status	The success or failure of the match attempt. **F** Failure (no DPID or CCD found) **null** Success
Status.Code	Reason for failure, if there is one. • UnableToValidate • InsufficientInputData
Status.Description	A description of the problem, if there is one.
AMAS.ResultCode	The result code returned by the underlying engine.
AMAS.ResultMessage	Any result messages returned by the underlying engine.

Result Codes

This option outputs result fields that are associated with certain output elements, as well as a result code for each result field, if available. If a result field does not have an accompanying result code, it may indicate one of the following:

- No change was made to the parsed element
- The parsed element was standardized (e.g., 'Street' changed to 'ST')
- No data was parsed into a corresponding parsed address element

Table 263: Result Codes

columnName	Result Code	
City.Result	**C**	Corrected
HouseNumber.Result	**U**	Unmatched, missing, or ambiguous
PostalCode.Result	**C**	Corrected
PostalDelivery.Result	**C**	Corrected
	D	Dropped
	U	Unmatched
StateProvince.Result	**C**	Corrected
StreetName.Result	**C**	Corrected
	U	Unmatched, missing, or ambiguous
StreetSuffix.Result	**C**	Corrected

Parsed Address Elements

This option outputs parsed address elements.

Table 264: Parsed Address Elements

columnName	Description
ApartmentLabel	The flat or unit type (such as STE or APT), for example: 123 E Main St **Apt** 3
ApartmentNumber	The flat or unit number, for example: 123 E Main St Apt **3**
FloorLabel	The floor/level type, for example: 123 E Main St Apt 3, 4th **Floor**
FloorNumber	The floor/level number, for example: 123 E Main St Apt 3, **4th** Floor

columnName	Description
LotNumber	The lot number, for example: Lot 7 Caldwell Hwy
PostalDeliveryLabel	The postal delivery type, for example: PO Box 42
PostalDeliveryNumber	The postal delivery number, for example: PO Box 42
PostalDeliveryPrefix	The postal delivery number prefix, for example: PO Box A42
PostalDeliverySuffix	The postal delivery number suffix, for example: PO Box 42B
HouseNumber	The house number 1, for example: 298A-1B New South Head Rd
HouseSuffix	The house number 1 suffix, for example: 298A-1B New South Head Rd
HouseNumber2	The house number 2, for example: 298A-1B New South Head Rd
HouseSuffix2	The house number 2 suffix, for example: 298A-1B New South Head Rd
StreetName	The name of street where property is located, for example: 123 E **Main** St Apt 3
StreetSuffix	The street suffix, for example: 123 E Main **St** Apt 3
TrailingDirectional	The trailing directional, for example: 123 Pennsylvania Ave **NW**

Original Input Data

This option outputs the original input data in <FieldName>.Input fields.

Table 265: Input Data

columnName	Description
AddressLine1.Input	The first address line passed on input.
AddressLine2.Input	The second address line passed on input.
AddressLine3.Input	The third address line passed on input.
AddressLine4.Input	The fourth address line passed on input.

columnName	Description
City.Input	The city/locality/suburb name passed on input.
StateProvince.Input	The state passed on input.
PostalCode.Input	The postal code passed on input.

ValidateAddressGlobal

ValidateAddressGlobal provides enhanced address standardization and validation for addresses outside the U.S. and Canada. ValidateAddressGlobal can also validate addresses in the U.S. and Canada but its strength is validation of addresses in other countries. If you process a significant number of addresses outside the U.S. and Canada, you should consider using ValidateAddressGlobal.

ValidateAddressGlobal is part of the Universal Addressing Module. For more information on the Universal Addressing Module, including a listing of other components included, see **What Is the Universal Addressing Module?** on page 722.

ValidateAddressGlobal performs several steps to achieve a quality address, including transliteration, parsing, validation, and formatting.

Character Set Mapping and Transliteration

ValidateAddressGlobal handles international strings and their complexities. It uses fully Unicode enabled string processing which enables the transliteration of non-roman characters into the Latin character set and mapping between different character sets.

Character set mapping and transliteration features include:

- Support for over 30 different character sets including UTF-8, ISO 8859-1, GBK, BIG5, JIS, EBCDIC
- Proper "elimination" of diacritics according to language rules
- Transliteration for various alphabets into Latin Script
- Greek (BGN/PCGN 1962, ISO 843 - 1997)
- Cyrillic (BGN/PCGN 1947, ISO 9 - 1995)
- Hebrew
- Japanese Katakana, Hiragana and Kanji
- Chinese Pinyin (Mandarin, Cantonese)
- Korean Hangul

Address Parsing, Formatting, and Standardization

Restructuring incorrectly fielded address data is a complex and difficult task especially when done for international addresses. People introduce many ambiguities as they enter address data into computer systems. Among the problems are misplaced elements (such as company or personal names in street

address fields) or varying abbreviations that are not only language, but also country specific. ValidateAddressGlobal identifies address elements in address lines and assigns them to the proper fields. This is an important precursor to the actual validation. Without restructuring, "no match" situations might result.

Properly identified address elements are also important when addresses have to be truncated or shortened to fit specific field length requirements. With the proper information in the right fields, specific truncation rules can be applied.

- Parses and analyzes address lines and identifies individual address elements
- Processes over 30 different character sets
- Formats addresses according to the postal rules of the country of destination
- Standardizes address elements (such as changing AVENUE to AVE)

Global Address Validation

Address validation is the correction process where properly parsed address data is compared against reference databases supplied by postal organizations or other data providers. ValidateAddressGlobal validates individual address elements to check for correctness using sophisticated fuzzy matching technology and produces standardized and formatted output based on postal standards and user preferences. FastCompletion validation type can be used in quick address entry applications. It allows input of truncated data in several address fields and generates suggestions based on this input.

In some cases, it is not possible to fully validate an address. Here ValidateAddressGlobal has a unique deliverability assessment feature that classifies addresses according to their probable deliverability.

Input

ValidateAddressGlobal takes a standard address as input. All addresses use this format no matter what country the address is from.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section

Table 266: ValidateAddressGlobal Input

columnName	Format	Description / Valid Values
AddressLine1 through AddressLine6	String [79]	These fields contain address line data. AddressLine1 contains the first address line, AddressLine2 contains the second address line, and so forth. Note that the city, state/province, and postal code information should be placed in their respective fields, not address line fields. For example: **AddressLine1:** 17413 Blodgett Road **AddressLine2:** PO Box 123 **City:** Mount Vernon **StateProvice:** WA

columnName	Format	Description / Valid Values
		PostalCode: 97273 **Country:** USA If the input address is not already parsed into the appropriate address line and City, StateProvice, and PostalCode fields, use the UnformattedLine fields instead of the address line fields.
City	String [79]	City name
StateProvince	String [79]	State or province.
PostalCode	String [79]: 99999 ~~99999999~~ A9A9A9 A9A 9A9 9999 999	The postal code for the address. In the U.S. this is the ZIP Code®.
Contact	String [79]	The name of the addressee. For example, "Mr. Jones".
Country	String [79]	The country name. If no value is specified in the `Input.ForceCountryISO3` or `Input.DefaultCountryISO3` option, you must specify a country.
FirmName	String [79]	Company or firm name
Street	String [79]	Street
Number	Building [79]	Number
Building	String [79]	Building

columnName	Format	Description / Valid Values
SubBuilding	String [79]	SubBuilding
DeliveryService	String [79]	DeliveryService
UnformattedLine1 through UnformattedLine10	String [79]	Use these fields if the input address is completely unparsed and you want ValidateAddressGlobal to attempt to parse the address into the appropriate fields. For example: **UnformattedLine1:** 17413 Blodgett Road **UnformattedLine2:** PO Box 123 **UnformattedLine3:** Mount Vernon WA 97273 **UnformattedLine4:** USA This address would be parsed into these output fields: **AddressLine1:** 17413 Blodgett Road **AddressLine2:** PO Box 123 **City:** Mount Vernon **StateProvice:** WA **PostalCode:** 97273 **Country:** USA **Note:** If you specify input in the unformatted line fields you must specify the entire address using only unformatted line fields. Do not use other fields such as City or StateProvince in combination with unformatted line fields.

Options

Input Options

Table 267: ValidateAddressGlobal Input Options

optionName	Description/Valid Values
Database.AddressGlobal	Specifies the database resource containing the postal data to use for address validation. Only databases that have been defined in the **Global Database Resources** panel in the Management Console are available. For more information, see the *Spectrum™ Technology Platform User's Guide*.

optionName	Description/Valid Values
Input.DefaultCountryISO3	Specifies a default country to use when the input record does not contain explicit country information. Specify the country using the ISO3 country code. If you do not specify a default country each input record must have the country specified in the Country input field. For a listing, see **Country ISO Codes and Module Support** on page 908.
Input.ForceCountryISO3	Causes address records to be always treated as originating from the country specified here, overriding the country in the address record and the default country. Specify the country using the ISO3 country code. For a listing, see **Country ISO Codes and Module Support** on page 908.

Output Options

Table 268: ValidateAddressGlobal Output Options

optionName	Description/Valid Values
Result.MaximumResults	This option specifies the maximum number of candidate addresses to return. The default is 1. The maximum is 20. If you are using FastCompletion mode, you may want to enter a number greater than 1 to ensure you are provided with multiple options for completing a field.
Result.IncludeInputs	Specifies whether to include the input data in the output. If enabled, the output will contain fields that end with .Input containing the corresponding input field. For example, the output field AddressLine1.Input would contain the data specified in the input field AddressLine1. One of the following: **TRUE** Include the input data in the output. **FALSE** Do not include the input data in the output (default).
Result.StateProvinceType	Specifies the format for the StateProvince field. One of the following. **ABBREVIATION** Return the abbreviation for the state or province. For example, North Carolina would be returned as "NC". **COUNTRY_STANDARD** Return either the abbreviation or the full name depending on the format used by the country's postal authority. (Default)

optionName	Description/Valid Values	
	EXTENDED	Return the full name of the state or province, not the abbreviation. For example "North Carolina".
Result.CountryType	Specifies the language or code to use for the country name returned by ValidateAddressGlobal.	
	ISO2	The two-character ISO code for the country
	ISO3	The three-character ISO code for the country
	ISO_NUMBER	The ISO country number
	NAME_CN	Chinese
	NAME_DA	Danish
	NAME_DE	German
	NAME_EN	English (default)
	NAME_ES	Spanish
	NAME_FI	Finnish
	NAME_FR	French
	NAME_GR	Greek
	NAME_HU	Hungarian
	NAME_IT	Italian
	NAME_JP	Japanese
	NAME_KR	Korean
	NAME_NL	Dutch
	NAME_PL	Polish
	NAME_PT	Portuguese
	NAME_RU	Russian
	NAME_SA	Sanskrit
	NAME_SE	Swedish
Result.PreferredScript	Specifies the alphabet in which the output should be returned. The alphabet in which the data is returned differs from country to country.	

optionName	Description/Valid Values
	For most countries the output will be Latin I regardless of the selected preferred language. **ASCII_Extended** — ASCII characters with expansion of special characters (e.g. Ã– = OE) **ASCII_Simplified** — ASCII characters **Database** — (default) Latin I or ASCII characters (as per reference database standard) **Latin** — Latin I characters **Latin_Alt** — Latin I characters (alternative transliteration) **Postal_Admin_Alt** — Latin I or ASCII characters (local postal administration alternative) **Postal_Admin_Pref** — Latin I or ASCII characters (as preferred by local postal administration) For countries that use an alphabet other than Latin I, the returned alphabet differs from country to country. For more information, see **Alphabets for Non-Latin 1 Countries** on page 847.
Result.PreferredLanguage	Specifies the language in which the output should be returned. The alphabet in which the data is returned differs from country to country, but for most countries the output will be Latin, regardless of the selected preferred language. **DATABASE** — Language derived from reference data for each address. Default. **ENGLISH** — English locality and state/province names output, if available.
Result.Casing	Specifies the casing of the output. **NATIVE** — Output will be based on the reference database standard. **UPPER** — Output will be in upper case for all countries. **LOWER** — Output will be in lower case for all countries. **MIXED** — Casing determined by country-specific rules. **NOCHANGE** — For parse mode, returns the data the way it was entered. For validation mode, uses the casing found in the reference data and according to postal rules.

optionName	Description/Valid Values
	Values that could not be checked against the reference data will retain their input casing.

Alphabets for Non-Latin 1 Countries

For countries that use an alphabet other than Latin I, the returned alphabet differs from country to country. The following table shows how the output is returned for specific countries. All countries that are not listed use the value specified in the field `Result.PreferredScript` option.

Country	Database	Postal_Admin_Ref	Postal_Admin_Alt	Latin	Latin_Alt	ASCII_Simplified	ASCII_Extended
RUS	Cyrillic	Cyrillic	Cyrillic	CYRILLIC_ISO	CYRILLIC_BGN	CYRILLIC_ISO + LATIN_SIMPLE	CYRILLIC_ISO + LATIN
JPN	Kanji	Kanji	Kana	JAPANESE	JAPANESE	JAPANESE + LATIN_SIMPLE	JAPANESE + LATIN
CHN	Hanzi	Hanzi	Hanzi	CHINESE_ MANDARIN	CHINESE_ CANTONESE	CHINESE_ MANDARIN + LATIN_SIMPLE	CHINESE_ MANDARIN + LATIN
HKG	Hanzi	Hanzi	Hanzi	CHINESE_ CANTONESE	CHINESE_ MANDARIN	CHINESE_ CANTONESE + LATIN_SIMPLE	CHINESE_ CANTONESE + LATIN
TWN	Hanzi	Hanzi	Hanzi	CHINESE_ CANTONESE	CHINESE_ MANDARIN	CHINESE_ CANTONESE + LATIN_SIMPLE	CHINESE_ CANTONESE + LATIN
GRC	Greek	Greek	Greek	GREEK_ISO	GREEK_BGN	GREEK_ISO + LATIN_SIMPLE	GREEK_ISO + LATIN
KOR	Latin	Hangul	Hanja	KOREAN	KOREAN	KOREAN + LATIN_SIMPLE	KOREAN + LATIN
ISR	Latin	Hebrew	Hebrew	HEBREW	HEBREW	HEBREW + LATIN_SIMPLE	HEBREW + LATIN
ROM	Latin-3	Latin-3	Latin-3	Latin-3	Latin-3	LATIN_SIMPLE	LATIN
POL	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
CZE	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
CRI	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN

Country	Database	Postal_Admin_Ref	Postal_Admin_Alt	Latin	Latin_Alt	ASCII_Simplified	ASCII_Extended
HUN	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
MDA	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
SVK	Latin-2	Latin-2	Latin-2	Latin-2	Latin-2	LATIN_SIMPLE	LATIN
LAT	Latin-7	Latin-7	Latin-7	Latin-7	Latin-7	LATIN_SIMPLE	LATIN

Process Options

Table 269: ValidateAddressGlobal Process Options

optionName	Description/Valid Values
Process.OptimizationLevel	Use this option to set the appropriate balance between processing speed and quality. One of the following: **NARROW** The parser will honor input assignment strictly, with the exception of separation of House Number from Street information. **STANDARD** The parser will separate address element more actively as follows: • Province will be separated from Locality information • PostalCode will be separated from Locality information • House Number will be separated from Street information • SubBuilding will be separated from Street information • DeliveryService will be separated from Street information • SubBuilding will be separated from Building information • Locality will be separated from PostalCode information **WIDE** Parser separation will happen similarly to Standard, but additionally up to 10 parsing candidates will be passed to validation for processing. Validation will widen its search tree and take additional reference data entries into account for matching. Please note that adjusting the optimization level might have no effect for countries that lack the postal reference data information required for the kind of separation described above. Increasing separation granularity from Narrow to Standard consumes some processing power, but the major impact on processing speed is from validation processing a larger search tree, thus increasing the number of data accesses and comparisons for the optimization level Wide, in an attempt to make the most out of the input data given.

optionName	Description/Valid Values	
Process.Mode	Specifies the type of processing to perform on the addresses. One of the following:	
	BATCH	Use this mode in batch processing environments when no human input or selection is possible. It is optimized for speed and will terminate its attempts to correct an address when ambiguous data is encountered that cannot be corrected automatically. The Batch processing mode will fall back to Parse mode when the database is missing for a specific country.
	CERTIFIED	Use this mode in batch processing environments for Australian mail. Validate Address Global is certified by Australia Post's Address Matching Approval System (AMAS). It will standardize and validate your mail against the Postal Address File, providing postal discounts and allowing for the least amount of undeliverable pieces.
	FASTCOMPLETION	Use this mode if you want to use FastCompletion mode to enter truncated data in address fields and have Validate Address Global generate suggestions. For example, if you work in a call center or point-of-sale environment, you can enter just part of an address element and the FastCompletion feature will provide valid options for the complete element.
	INTERACTIVE	Use this mode when working in interactive environments to generate suggestions when an address input is ambiguous. This validation type is especially useful in data entry environments when capturing data from customers or prospects. It requires the input of an almost-complete address and will attempt to validate or correct the data provided. If ambiguities are detected, this validation type will generate up to 20 suggestions that can be used for pick lists. The Interactive processing mode will fall back to Parse mode when the respective database is missing for a specific country.
	PARSE	Use this mode for separating address input into tokens for subsequent processing in other systems, bypassing validation. For example, you could use this mode when address data of already high quality simply needs to be tokenized quickly for export to an external system or for use by a downstream stage.

optionName	Description/Valid Values
Process.MatchingScope	Specifies how closely an address must match the reference data in order for the address to be validated. One of the following: **Note:** These settings may not have an effect for countries lacking the necessary level of detail in the postal reference data. **ALL** — All address elements must match. **DELIVERYPOINT_LEVEL** — Validate Global Address must achieve a match on StateProvince, PostalCode, City/Locality/Suburb, street, house number, and sub building. **STREET_LEVEL** — Validate Global Address must achieve a match on StateProvince, PostalCode, City/Locality/Suburb, and street. **LOCALITY_LEVEL** — Validate Global Address must achieve a match on StateProvince, PostalCode, and City/Locality/Suburb.

Output

Address Data

Table 270: Parsed Address Elements

columnName	Description
AddressBlock1-9	The AddressBlock output fields contain a formatted version of the standardized or normalized address as it would be printed on a physical mailpiece. Validate Address Global formats the address into address blocks using postal authority standards. Each line of the address is returned in a separate address block field. There can be up to nine address block output fields: AddressBlock1 through AddressBlock9. For example, this input address: AddressLine1: 4200 Parliament Place AddressLine2: Suite 600 City: Lanham StateProvince: MD PostalCode: 20706 Results in this address block output:

columnName	Description
	AddressBlock1: 4200 PARLIAMENT PL STE 600 AddressBlock2: LANHAM MD 20706-1882
AddressLine1-6	If the address was validated, the address line fields contain the validated and standardized address lines. If the address could not be validated, the address line fields contain the input address without any changes. Note that the last line of the address is contained in the LastLine field. For example: AddressLine1: 4200 PARLIAMENT PL STE 600 LastLine: LANHAM MD 20706-1882
AdministrativeDistrict	An area smaller than a state/province but larger than a city.
ApartmentLabel	The flat or unit type (such as STE or APT), for example: 123 E Main St **Apt** 3
ApartmentNumber	The flat or unit number, for example: 123 E Main St Apt **3**
BlockName	An estate or block name.
BuildingName	The name of a building, for example Sears Tower.
City	The name of the town or city. For example, **Vancouver**, BC.
City.AddInfo	Additional information about the city.
City.SortingCode	A code used by the postal authority to speed up delivery in certain countries for large localities, for example Prague or Dublin.
Contact	The name of the addressee. For example, **Mr. Jones**.
Country	The country in the language or code specified in the `Result.CountryType` option.
County	Dependent state or province information that further subdivides a state or province. An example would be a U.S. county.
FirmName	The name of a company.
Floor	Information that further subdivides a building, e.g. the suite or apartment number. For example: 123 E Main St Apt 3, **4th Floor**
HouseNumber	The house number 1, for example: 298A-1B New South Head Rd
LastLine	Complete last address line (city, state/province, and postal code).
LeadingDirectional	Street directional that precedes the street name. For example, the N in 138 N Main Street.

columnName	Description
Locality	Dependent place name that further subdivides a Locality. Examples are colonias in Mexico, Urbanisaciones in Spain.
POBox	Post Box descriptor (POBox, Postfach, Case Postale etc.) and number.
PostalCode	The postcode for the address. The format of the postcode varies by country.
PostalCode.AddOn	The second part of a postcode. For example, for Canadian addresses this will be the LDU. For U.S. addresses this is the ZIP + 4 add on. This field is not used by most countries.
PostalCode.Base	The base portion of the postcode.
Room	A room number in a building.
SecondaryStreet	The name of a secondary street or rural route.
StateProvince	The name of the state or province.
StreetName	The name of street where property is located, for example: 123 E **Main** St Apt 3
StreetSuffix	The street suffix, for example: 123 E Main **St** Apt 3
SubBuilding	A portion of a building, such as a suite. For example, Suite 102.
Suburb	Dependent place name that further subdivides a Locality. An example would be Mahalle in Turkey.
Territory	The name of a territory. Territories are larger than a state/province.
TrailingDirectional	The trailing directional, for example: 123 Pennsylvania Ave **NW**

Original Input Data

This option outputs the original input data in <FieldName>.Input fields.

Table 271: Original Input Data

columnName	Format	Description / Valid Values
AddressLine1.Input	String [79]	First address line
AddressLine2.Input	String [79]	Second address line

columnName	Format	Description / Valid Values
AddressLine3.Input	String [79]	Third address line
AddressLine4.Input	String [79]	Fourth address line
AddressLine5.Input	String [79]	Fifth address line
AddressLine6.Input	String [79]	Sixth address line
City.Input	String [79]	City name
StateProvince.Input	String [79]	State or province
PostalCode.Input	String [79]:	The postal code for the address. In the U.S. this is the ZIP Code. One of these formats: 99999 99999-9999 A9A9A9 A9A 9A9 9999 999
Contact.Input	String [79]	The name of the addressee. For example, "Mr. Jones".
Country.Input	String [79]	Specify the country using the format you chose for input country format (English name, ISO code, or UPU code). For a list of valid values, see **Country ISO Codes and Module Support** on page 908.
FirmName.Input	String [79]	Company or firm name
Street.Input	String [79]	Street
Number.Input	Building [79]	Number

columnName	Format	Description / Valid Values
Building.Input	String [79]	Building
SubBuilding.Input	String [79]	SubBuilding
DeliveryService.Input	String [79]	DeliveryService

Result Codes

These output fields contain information about the result of the validation processing.

Table 272: Result Codes

columnName	Result Code
AddressType	For United States and Canada addresses only, the AddressType field indicates the type of address. One of the following: **F** The address was validated/corrected to the firm name. **B** The address was validated/corrected to the building name. **G** The address is a general delivery address. **H** The address was validated/corrected to the high-rise default. **L** The address is a large volume receiver. **M** The address is a military address. **P** The address was validated/corrected to PO box. **R** The address was validated/corrected to a rural route. **S** The address was validated/corrected to a street address. **U** The address could not be validated/corrected so the type is unknown.
Confidence	The level of confidence assigned to the address being returned. Range is from zero (0) to 100; zero indicates failure, 100 indicates a very high level of confidence that the match results are correct.
CountOverflow	Indicates whether the number of candidate addresses exceeds the number returned. One of the following:

columnName	Result Code	
	Yes	Yes, there are additional candidate addresses. To obtain the additional candidates, increase the `MaximumResults` value.
	No	No, there are no additional candidates.
ElementInputStatus	ElementInputStatus provides per element information on the matching of input elements to reference data. The values in this field vary depending on whether you are using batch mode or parse mode. For information about the value in this field, see **Interpreting ElementInputStatus, ElementResultStatus, and ElementRelevance** on page 858.	
ElementRelevance	Indicates which address elements are actually relevant from the local postal authority's point of view. For information about the value in this field, see **Interpreting ElementInputStatus, ElementResultStatus, and ElementRelevance** on page 858.	
ElementResultStatus	ElementResultStatus categorizes the result in more detail than the ProcessStatus field by indicating if and how the output fields have been changed from the input fields. For information about the value in this field, see **Interpreting ElementInputStatus, ElementResultStatus, and ElementRelevance** on page 858.	
MailabilityScore	An estimate of how likely it is that mail sent to the address would be successful delivered. One of the following:	
	5	Completely confident of deliverability
	4	Almost certainly deliverable
	3	Should be deliverable
	2	Fair chance
	1	Risky
	0	No chance
ModeUsed	Indicates the processing mode used. The processing mode is specified in the `Process.Mode` option. For a description of the modes, see **Process Options** on page 848.	
MultimatchCount	If the address was matched to multiple candidate addresses in the reference data, this field contains the number of candidate matches found.	
ProcessStatus	Provides a general description of the output quality. For a more detailed description of the output quality, see the ElementResultStatus field. One of the following:	

columnName	Result Code	
	V4	Verified. The input data is correct. All elements were checked and input matched perfectly.
	V3	Verified. The input data is correct on input but some or all elements were standardized or the input contains outdated names or exonyms.
	V2	Verified. The input data is correct but some elements could not be verified because of incomplete reference data.
	V1	Verified. The input data is correct but the user standardization has deteriorated deliverability (wrong element user standardization - for example, postcode length chosen is too short). Not set by validation.
	C4	Corrected. All elements have been checked.
	C3	Corrected, but some elements could not be checked.
	C2	Corrected, but delivery status unclear (lack of reference data).
	C1	Corrected, but delivery status unclear because user standardization was wrong. Not set by validation.
	I4	Data could not be corrected completely, but is very likely to be deliverable. Single match (e.g. HNO is wrong but only 1 HNO is found in reference data).
	I3	Data could not be corrected completely, but is very likely to be deliverable. Multiple matches (e.g. HNO is wrong but more than 1 HNO is found in reference data).
	I2	Data could not be corrected, but there is a slim chance that the address is deliverable.
	I1	Data could not be corrected and is unlikely to be delivered.
	RA	Country recognized from the Force country Setting
	R9	Country recognized from DefaultCountryISO3 Setting
	R8	Country recognized from name without errors
	R7	Country recognized from name with errors
	R6	Country recognized from territory
	R5	Country recognized from province
	R4	Country recognized from major town
	R3	Country recognized from format
	R2	Country recognized from script

columnName	Result Code	
	R1	Country not recognized - multiple matches
	R0	Country not recognized
	S4	Parsed perfectly
	S3	Parsed with multiple results
	S2	Parsed with errors. Elements change position.
	S1	Parse Error. Input Format Mismatch.
	N1	Validation Error: No validation performed because country was not recognized.
	N2	Validation Error: No validation performed because required reference database is not available.
	N3	Validation Error: No validation performed because country could not be unlocked.
	N4	Validation Error: No validation performed because reference database is corrupt or in wrong format.
	N5	Validation Error: No validation performed because reference database is too old.
	N6	Validation Error: No validation performed because input data was insufficient.
	Q3	FastCompletion Status: Suggestions are available - complete address.
	Q2	FastCompletion Status: Suggested address is complete but combined with elements from the input (added or deleted).
	Q1	FastCompletion Status: Suggested address is not complete (enter more information).
	Q0	FastCompletion Status: Insufficient information provided to generate suggestions.
Status	Reports the success or failure of the processing attempt.	
	null	Success
	F	Failure
Status.Code	The reason for the failure, if there was one.	
Status.Description	A description of the reason for the failure, if there was one.	

Interpreting ElementInputStatus, ElementResultStatus, and ElementRelevance

The ElementInputStatus, ElementResultStatus, and ElementRelevance output fields contain a series of digits that describe the outcome of the validation operation in detail. ElementInputStatus contains some information for parsing operations.

This is what an ElementInputStatus value looks like:

44606040600000000060

This is what an ElementResultStatus value looks like:

88F0F870F00000000040

This is what an ElementRelevance value looks like:

11101010100000000000

To understand the values in these fields you need to know which element each position represents, and the meaning of the values in each position. For example, the first digit indicates the result from the PostalCode.Base output field. The position meanings are listed below.

- Position 1—PostalCode.Base
- Position 2—PostalCode.AddOn
- Position 3—City
- Position 4—Locality and Suburb
- Position 5—StateProvice
- Position 6—County
- Position 7—StreetName
- Position 8—SecondaryStreet
- Position 9—HouseNumber
- Position 10—Number level 1
- Position 11—POBox
- Position 12—Delivery service level 1
- Position 13—Building level 0
- Position 14—BuildingName
- Position 15—Sub building level 0
- Position 16—Floor and Room
- Position 17—FirmName
- Position 18—Organization level 1
- Position 19—Country
- Position 20—Territory

For ElementInputStatus, the possible values for validation are:

- 0—Empty
- 1—Not found
- 2—Not checked (no reference data)
- 3—Wrong - Set by validation only: The reference database suggests that either Number or DeliveryService is out of valid number range. Input is copied, not corrected for batch mode, for interactive mode and FastCompletion suggestions are provided.

- 4—Matched with errors in this element
- 5—Matched with changes (inserts and deletes) For example:
  - Parsing: Splitting of house number for "MainSt 1"
  - Validation: Replacing input that is an exonym or dropping superfluous fielded input that is invalid according to the country reference database
- 6—Matched without errors

For ElementInputStatus, the possible values for parsing are:

- 0—Empty
- 1—Element had to be relocated
- 2—Matched but needed to be normalized
- 3—Matched

For ElementRelevance, the possible values for parsing are:

- 0—Empty
- 1—Element had to be relocated
- 2—Matched but needed to be normalized
- 3—Matched

For ElementResultStatus, the possible values are (for all address elements apart from country):

- 0—Empty
- 1—Not validated and not changed. Original is copied.
- 2—Not validated but standardized.
- 3—Validated but not changed due to invalid input, database suggests that number is out of valid ranges. Input is copied, not corrected - this status value is only set in batch mode.
- 4—Validated but not changed due to lack of reference data.
- 5—Validated but not changed due to multiple matches. Only set in batch mode, otherwise multiple suggestions that replace the input are marked as corrected (status value 7).
- 6—Validated and changed by eliminating the input value
- 7—Validated and changed due to correction based on reference data
- 8—Validated and changed by adding value based on reference data
- 9—Validated, not changed, but delivery status not clear (e.g. DPV value wrong; given number ranges that only partially match reference data).
- C—Validated, verified but changed due to outdated name
- D—Validated, verified but changed from exonym to official name
- E—Validated, verified but changed due to standardization based on casing or language. Validation only sets this status if input fully matches a language alternative.
- F—Validated, verified and not changed due to perfect match

For Country (position 19 & 20), the following values are possible:

- 0—Empty
- 1—Country not recognized
- 4—Country recognized from DefaultCountryISO3 setting

- 5—Country not recognized - multiple matches
- 6—Country recognized from script
- 7—Country recognized from format
- 8—Country recognized from major town
- 9—Country recognized from province
- C—Country recognized from territory
- D—Country recognized from name with errors
- E—Country recognized from name without errors
- F—Country recognized from ForceCountryISO3 setting

Reports

Validate Address Global Summary Report

Job Summary

This section contains summary information about the job.

- **Started**—The date and time that the job started.
- **Finished**—The date and time that the job ended.
- **Processing time**—The duration of the job.
- **Total Records**—The total number of records presented to Validate Address Global for processing. This may be different from the number of input records for the job depending on how the job is designed.
- **Processed Records**—The number of addresses that were successfully processed by Validate Address Global. This is the total number of records less records not processed.
- **Default country**—The default country specified in the **Default country (ISO3 format)** option.
- **Casing**—The casing selected in the **Casing** option.
- **Script/Alphabet**—The script specified in the **Script/Alphabet** option.
- **Countries**—The number of countries represented in the input addresses.

Status Summary

This section lists the validation and correction results.

- **Validated**—Addresses that were correct on input.
- **Corrected**—Addresses that were corrected by Validate Address Global.
- **Good deliverability**—Addresses that could not be corrected but that are very likely to be deliverable.
- **Fair deliverability**—Addresses that could not be corrected but have a fair chance that the address is deliverable.
- **Poor deliverability**—Addresses that could not be corrected and are unlikely to be deliverable.
- **Parsed**—Addresses that were successfully parsed.
- **Failed**—Addresses that could not be verified, corrected, or parsed.

Validate Address Global Detail Report

Status Details

This section lists the validation and correction results for each country.

- **V (Validated)**—Addresses that were correct on input.
- **C (Corrected)**—Addresses that were corrected by Validate Address Global.
- **I4 (Good deliverability)**—Addresses that could not be corrected but that are very likely to be deliverable.
- **I3 (Fair deliverability)**—Addresses that could not be corrected but have a fair chance that the address is deliverable.
- **I2 (Poor deliverability)**—Addresses that could not be corrected and are unlikely to be deliverable.
- **S (Parsed)**—Addresses that were successfully parsed.
- **F (Failed)**—Addresses that could not be verified, corrected, or parsed.

ValidateAddressLoqate

ValidateAddressLoqate standardizes and validates addresses using postal authority address data. ValidateAddress Loqate can correct information and format the address using the format preferred by the applicable postal authority. It also adds missing postal information, such as postal codes, city names, state/province names, and more.

ValidateAddressLoqate also returns result indicators about validation attempts, such as whether or not ValidateAddressLoqate validated the address, the level of confidence in the returned address, the reason for failure if the address could not be validated, and more.

During address matching and standardization, ValidateAddressLoqate separates address lines into components and compares them to the contents of the Universal Addressing Module databases. If a match is found, the input address is *standardized* to the database information. If no database match is found, ValidateAddressLoqate optionally *formats* the input addresses. The formatting process attempts to structure the address lines according to the conventions of the appropriate postal authority.

ValidateAddressLoqate is part of the Universal Addressing Module. For more information on this module, including a listing of other components included with it, see **What Is the Universal Addressing Module?** on page 722

Input

ValidateAddressLoqate takes an address as input. All addresses use this format regardless of the address's country. See **Address Line Processing for U.S. Addresses** on page 862 for important information about how address line data is processed for U.S. addresses.

Note: You specify input using the DataTable class. For information on the DataTable class, see the "API Fundamentals" section.

Table 273: Input Format

columnName	Format	Description / Valid Values
AddressLine1	String	The first address line.
AddressLine2	String	The second address line.
AddressLine3	String	The third address line.
AddressLine4	String	The fourth address line.
City	String	The city name.
Country	String	The country code or name, in any of the following formats: • Two-character ISO 3116-1 Alpha-2 country code • Three-character ISO 3116-1 Alpha-3 country code • English country name For a listing of ISO codes, see **Country ISO Codes and Module Support** on page 908.
FirmName	String	The company or firm name.
PostalCode	String	The postal code for the address in one of the following formats: 99999 99999-9999 A9A9A9 A9A 9A9 9999 999
StateProvince	String	The state or province.

Address Line Processing for U.S. Addresses

The input fields AddressLine1 through AddressLine4 are handled differently for U.S. addresses depending on whether the firm name extraction or urbanization code extraction options are enabled. If either of these options is enabled, ValidateAddressLoqate will look at the data in all four fields to validate the address and extract the requested data (firm name and/or urbanization code). If neither of these options is enabled, ValidateAddressLoqate uses only the first two non-blank address line fields in its validation attempt. The data in the other address line fields is returned in the output field AdditionalInputData. For example,

AddressLine1: A1 Calle A
AddressLine2:
AddressLine3: URB Alamar
AddressLine4: Pitney Bowes Software

In this address, if either firm name extraction or urbanization code extraction were enabled, ValidateAddressLoqate would examine all four address lines. If neither firm name extraction nor urbanization code extraction were enabled, ValidateAddressLoqate would examine AddressLine1 and AddressLine3 (the first two non-blank address lines) and attempt to validate the address using that data; the data in AddressLine4 would be returned in the output field AdditionalInputData.

Options

The following table lists the options that control the type of information returned by ValidateAddressLoqate.

Table 274: Output Data Options

optionName	Description/Valid Values
Database.Loqate	Specifies which database you want to use for validating international addresses. To specify a database for international address validation, select a database in the **Database** drop-down list.
OutputFieldLevelReturnCodes	Specifies whether to include field-level result indicators. Field-level result indicators describe how ValidateAddressLoqate handled each address element. Field-level result indicators are returned in the qualifier "Result". For example, the field-level result indicator for HouseNumber is contained in **HouseNumber.Result**. For a complete listing of result indicator output fields, see **Result Indicators** on page 874. **N** No, do not output field-level return codes (default). **Y** Yes, output field-level return codes.
OutputFormattedOnFail	Specifies whether to return a formatted address when an address cannot be validated. The address is formatted using the preferred address format for the address's country. If this option is not selected, the output address fields are blank when ValidateAddressLoqate cannot validate the address. **Note:** This option applies only to U.S. and Canadian addresses. Formatted data will not be returned for any other address. **N** No, do not format failed addresses (default).

optionName	Description/Valid Values
	Y Yes, format failed addresses. Formatted addresses are returned using the format specified by the**OutputRecordType** option. Note that if you specify **OutputRecordType=E**, the parsed address elements will contain the parsed, validated address for addresses that could be validated. If the address could not be validated the parsed address elements will contain the input address in parsed form. If you always want the output to contain the input address in parsed form, regardless of whether or not ValidateAddressLoqate could validate the address, specify **OutputRecordType=I**. If you specify Y, you must specify "A" and/or "E" for OutputRecordType.
OutputAddressBlocks	Specifies whether to return a formatted version of the address as it would be printed on a physical mailpiece. Each line of the address is returned in a separate address block field. There can be up to nine address block output fields: **AddressBlock1** through **AddressBlock9**. For example, this input address: AddressLine1: 4200 Parliament Place AddressLine2: Suite 600 City: Lanham StateProvince: MD PostalCode: 20706 Results in this address block output: AddressBlock1: 4200 PARLIAMENT PL STE 600 AddressBlock2: LANHAM MD 20706-1882 AddressBlock3: UNITED STATES OF AMERICA ValidateAddressLoqate formats the address into address blocks using postal authority standards. The country name is returned using the Universal Postal Union country name. Note that the option OutputCountryFormat does not affect the country name in the address block, it only affects the name returned in the **Country** output field. For addresses outside the U.S. and Canada, if ValidateAddressLoqate is unable to validate the address, no address blocks are returned. For addresses in the U.S. and Canada, address blocks are returned even if validation fails. One of the following:

optionName	Description/Valid Values
	N No, do not return address blocks. Default. **Y** Yes, return address blocks.
OutputCasing	Specifies the casing of the output data. One of the following: **M** The output in mixed case (default). For example: 123 Main St Mytown FL 12345 **U** The output in upper case. For example: 123 MAIN ST MYTOWN FL 12345
HomeCountry	Specifies the default country. You should specify the country where most of your addresses reside. For example, if most of the addresses you process are in Canada, specify Canada. ValidateAddressLoqate uses the country you specify to attempt validation when it cannot determine the country from the StateProvince, PostalCode, and Country address fields. The valid country names are: Afghanistan, Albania, Algeria, American Somoa, Andorra, Angola, Anguilla, Antigua And Barbuda, Argentina, Armenia, Aruba, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bermuda, Bhutan, Bolivia, Bosnia And Herzegovina, Botswana, Brazil, British Virgin Islands, Brunei Darussalam, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon, Canada, Cape Verde, Cayman Islands, Central African Republic, Chad, Chile, China, Colombia, Comoros Islands, Congo, Cook Islands, Costa Rica, Cote D'Ivoire, Croatia, Cuba, Cyprus, Czech Republic, Democratic Republic Of Congo, Denmark, Djibouti, Dominica, Dominican Republic, East Timor, Ecuador, Egypt, El Salvador, Equitorial Guinea, Eritrea, Estonia, Ethiopia, Falkland Islands, Faroe Islands, Federated States Of Micronesia, Fiji, Finland, France, French Guiana, Gabon, Gambia, Germany, Ghana, Gibraltar, Greece, Greenland, Grenada, Guadeloupe, Guam, Guatemala, Guinea, Guinea Bissau, Guyana, Haiti, Holy See, Honduras, Hong Kong, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Korea, Kuwait, Kyrgyzstan, Laos, Latvia,

optionName	Description/Valid Values
	Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania, Luxembourg, Macau, Macedonia, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall Islands, Martinique, Mauritania, Mauritius, Mayotte, Mexico, Moldova, Monaco, Mongolia, Monserrat, Morocco, Mozambique, Myanmar, Namibia, Nauru, Nepal, Netherlands Antilles, New Caledonia, New Zealand, Nicaragua, Niger, Nigeria, Niue, Norway, Oman, Pakistan, Palau, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Pitcairn Islands, Poland, Portugal, Puerto Rico, Qatar, Republic Of Georgia, Republic Of Korea, Republic Of Singapore, Reunion, Romania, Russia, Rwanda, Saint Helena, Saint Kitts And Nevis, Saint Lucia, Saint Pierre And Miquelon, Saint Vincent And The Grenadines, Samoa, San Marino, Sao Tome And Principe, Saudi Arabia, Senegal, Seychelles, Sierra Leone, Slovakia, Slovenia, Solomon Islands, Somalia, South Africa, Spain, Sri Lanka, Sudan, Surivalue, Swaziland, Sweden, Switzerland, Syria, Tahiti, Taiwan, Tajikistan, Tanzania, Thailand, The Netherlands, Togo, Tonga, Trinidad And Tobago, Tristan Da Cunha, Tunisia, Turkey, Turkmenistan, Turks And Caicos Islands, Tuvalu, Uganda, Ukraine, United Arab Emirates, United Kingdom, United States, Uruguay, Uzbekistan, Vanuatu, Venezuela, Vietnam, Virgin Islands (US), Wallis And Futuna, Yemen, Yugoslavia, Zambia, Zimbabwe
OutputCountryFormat	Specifies the format to use for the country name returned in the **Country** output field. For example, if you select English, the country name "Deutschland" would be returned as "Germany". **E** Use English country names (default). **I** Use two-letter ISO abbreviation for the countries instead of country names. **U** Use Universal Postal Union abbreviation for the countries instead of country names.
OutputScript/Alphabet	Specifies the alphabet or script in which the output should be returned. This option is bi-directional and generally takes place from Native to Latin and Latin to Native. **Input** Do not perform transliteration and provide output in the same script as the input (default).

optionName	Description/Valid Values	
	Native	Output in the native script for the selected country wherever possible.
	Latn	Use English values.
	Cyrl	Use Cyrillic values.
	Grek	Use Greek values.
	Hebr	Use Hebrew values.
	Hani	Use Kanji values.
	Hans	Use Simplified Chinese values.
	Arab	Use Arabic values.
	Thai	Use Thai values.
	Hang	Use Hangul values.
KeepMultimatch	Indicates whether or not to return multiple address for those input addresses that have more than one possible match.	
	Y	Yes, return multiple matches (default).
	N	No, do not return multiple matches.
	For more information, see **Returning Multiple Matches** on page 867.	
Data license error handling	Specifies how you want Spectrum Technology Platform to respond when a data license error occurs.	
	Fail the job	Fail the entire job if a data license error occurs.
	Fail the record	Fail the record(s) for which the data license error occurs and continue processing.

Returning Multiple Matches

If ValidateAddressLoqate finds multiple address in the postal database that are possible matches for the input address, you can have ValidateAddressLoqate return the possible matches. For example, the following address matches multiple addresses in the U.S. postal database:

PO BOX 1 New York, NY

Options

To return multiple matches, use the options described in the following table.

Table 275: Multiple Match Option

optionName	Description/Valid Values
KeepMultimatch	Indicates whether or not to return multiple address for those input addresses that have more than one possible match. **Y** Yes, return multiple matches (default). **N** No, do not return multiple matches.
MaximumResults	A number between 1 and 10 that indicates the maximum number of addresses to return. The default value is 1. **Note:** The difference between Keepmultimatch=N and KeepMultimatch=Y/MaximumResults=1 is that a multiple match will return a failure if KeepMultimatch=N, whereas a multiple match will return one record if KeepMultimatch=Y and MaximumResults=1.
OutputFieldLevelReturnCodes	To identify which output addresses are candidate addresses, you must specify a value of **Y** for OutputFieldLevelReturnCodes. When you do this, records that are candidate addresses will have one or more "M" values in the field-level result indicators.

Output

When you choose to return multiple matches, the addresses are returned in the address format you specify. For information on specifying address format, see **Options** on page 863. To identify which records are the candidate addresses, look for multiple "M" values in the field-level result indicators. For more information**Result Indicators** on page 874, see .

Output

The output from ValidateAddressLoqate contains various information depending on the output categories you select.

The output categories are:

- **Standard Address Output** on page 869
- **Parsed Address Elements Output** on page 870
- **Geocode Output** on page 873
- **Result Indicators** on page 874

Standard Address Output

Standard address output consists of four lines of the address which correspond to how the address would appear on an actual address label. City, state/province, postal code, and other data is also included in standard address output. ValidateAddressLoqate returns standard address output for validated addresses if you set **OutputRecordType = A**. Standard address fields are always returned for addresses that could not be validated regardless of whether or not you set **OutputRecordType = A**. For non-validated addresses, the standard address output fields contain the address as it appeared in the input ("pass through" data). If you want ValidateAddressLoqate to standardize address according to postal authority standards when validation fails, specify **OutputFormattedOnFail** = Y in your request.

Table 276: Standard Address Output

columnName	Description / Valid Values
AdditionalInputData	Input data that could not be matched to a particular address component. For more information, see **About Additional Input Data** on page 884.
AddressLine1-4	If the address was validated, the first line of the validated and standardized address. If the address could not be validated, the first line of the input address without any changes. There can be up to four address block output fields: AddressLine1 through AddressLine4.
City	The validated city name.
Country	The country in the format determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English
FirmName	The validated firm or company name.
PostalCode	The validated ZIP Code™ or postal code.
PostalCode.AddOn	The 4-digit add-on part of the ZIP Code™. For example, in the ZIP Code™ 60655-1844, 1844 is the 4-digit add-on.
PostalCode.Base	The 5-digit ZIP Code™; for example 20706.
StateProvince	The validated state or province abbreviation.

Parsed Address Elements Output

Output addresses are formatted in the parsed address format if you set **OutputRecordType** = **E**. If you want ValidateAddressLoqate to return formatted data in the Parsed Address format when validation fails (that is, a normalized address), specify **OutputFormattedOnFail** = Y.

Note: If you want ValidateAddressLoqate to always return parsed input data regardless of whether or not validation is successful, specify **OutputRecordType = I**. For more information, see **Parsed Input** on page 871.

Table 277: Parsed Address Output

columnName	Description / Valid Values
AddressBlock1-9	The AddressBlock output fields contain a formatted version of the standardized or normalized address as it would be printed on a physical mailpiece. Validate Address Global formats the address into address blocks using postal authority standards. Each line of the address is returned in a separate address block field. There can be up to nine address block output fields: AddressBlock1 through AddressBlock9. For example, this input address: AddressLine1: 4200 Parliament Place AddressLine2: Suite 600 City: Lanham StateProvince: MD PostalCode: 20706 Results in this address block output: AddressBlock1: 4200 PARLIAMENT PL STE 600 AddressBlock2: LANHAM MD 20706-1882
ApartmentLabel	Apartment designator (such as STE or APT), for example: 123 E Main St **APT** 3
ApartmentNumber	Apartment number. For example: 123 E Main St APT **3**
ApartmentNumber2	Secondary apartment number. For example: 123 E Main St APT 3, **4th** Floor **Note:** In this release, this field will always be blank.

columnName	Description / Valid Values
City	Validated city name
Country	Country. Format is determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English
FirmName	The validated firm or company name
HouseNumber	House number, for example: **123** E Main St Apt 3
LeadingDirectional	Leading directional, for example: 123 **E** Main St Apt 3
POBox	Post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode	Validated postal code. For U.S. addresses, this is the ZIP Code.
StateProvince	Validated state or province name
StreetName	Street name, for example: 123 E **Main** St Apt 3
StreetSuffix	Street suffix, for example: 123 E Main **St** Apt 3
TrailingDirectional	Trailing directional, for example: 123 Pennsylvania Ave **NW**

Parsed Input

The output can include the input address in parsed form. This type of output is referred to as "parsed input." Parsed input fields contain the address data that was used as input regardless of whether or not ValidateAddress validated the address. Parsed input is different from the "parsed address elements" output in that parsed address elements contain the validated address if the address could be validated, and, optionally, the input address if the address could not be validated. Parsed input always contains the input address regardless of whether or not ValidateAddress validated the address.

To include parsed input fields in the output, set **OutputRecordType** = **I**.

Table 278: Parsed Input

columnName	Description / Valid Values
ApartmentLabel.Input	Apartment designator (such as STE or APT), for example: 123 E Main St **APT** 3
ApartmentNumber.Input	Apartment number, for example: 123 E Main St APT **3**
City.Input	Validated city name
Country.Input	Country. Format is determined by what you selected in OutputCountryFormat: • ISO Code • UPU Code • English
FirmName.Input	The validated firm or company name
HouseNumber.Input	House number, for example: **123** E Main St Apt 3
LeadingDirectional.Input	Leading directional, for example: 123 **E** Main St Apt 3
POBox.Input	Post office box number. If the address is a rural route address, the rural route box number will appear here.
PostalCode.Input	Validated postal code. For U.S. addresses, this is the ZIP Code.
StateProvince.Input	Validated state or province name
StreetName.Input	Street name, for example: 123 E **Main St** Apt 3
StreetSuffix.Input	Street suffix, for example: 123 E Main St Apt 3
TrailingDirectional.Input	Trailing directional, for example: 123 Pennsylvania Ave **NW**

Geocode Output

ValidateAddressLoqate returns the latitude/longitude, geocoding match code, dependent and double dependent localities, dependent thoroughfare, subadministrative and superadministrative areas, and the search distance as output. Match codes describe how well the geocoder matched the input address to a known address; they also describe the overall status of a match attempt. Search distance codes represent how close the geocode is to the actual physical location of an address. The output returned is in the DataTable class. For information on the DataTable class, see the "API Fundamentals" section .

Table 279: Standard Address Output

columnName	Description / Valid Values
Geocode.MatchCode	This two-byte code reflects the status and level of geocode matching for an address. The first byte represents the geocoding status and is one of the following: **A** Multiple candidate geocodes were found to match the input address, and an average of these was returned **I** A geocode was able to be interpolated from the input addresses location in a range **P** A single geocode was found matching the input address **U** A geocode was not able to be generated for the input address The second byte represents the level of geocoding matching and is one of the following: **5** Delivery point (post box or subbuilding) **4** Premise or building **3** Thoroughfare **2** Locality **1** Administrative area **0** None
Latitude	Eight-digit number in degrees and calculated to five decimal places (in the format specified).
Longitude	Eight-digit number in degrees and calculated to five decimal places (in the format specified).

columnName	Description / Valid Values
SearchDistance	The radius of accuracy in meters, providing an indication of the probable maximum distance between the given geocode and the actual physical location. This field is derived from and dependent upon the accuracy and coverage of the underlying reference data.

Result Indicators

Result indicators provide information about the kinds of processing performed on an address. There are two types of result indicators:

- **Record-Level Result Indicators**
- **Field-Level Result Indicators**

Record-Level Result Indicators

Record-level result indicators provide data about the results of ValidateAddressLoqate processing for each record, such as the success or failure of the match attempt, which coder processed the address, and other details. The following table lists the record-level result indicators returned by ValidateAddressLoqate.

Table 280: Record Level Indicators

columnName	Description / Valid Values
Confidence	The level of confidence assigned to the address being returned. Range is from zero (0) to 100; zero indicates failure, 100 indicates a very high level of confidence that the match results are correct. For multiple matches, the confidence level is 0. For details about how this number is calculated, see **Introduction to the Validate Address Confidence Algorithm** on page 889.
CouldNotValidate	If no match was found, which address component could not be validated: • ApartmentNumber • HouseNumber • StreetName • PostalCode • City • Directional • StreetSuffix • Firm • POBoxNumber

columnName	Description / Valid Values
	Note: More than one component may be returned, in a comma-separated list.
MatchScore	MatchScore provides an indication of the similarity between the input data and the closest reference data match. It is significantly different from Confidence in that Confidence indicates how much the input address changed to obtain a match, whereas the meaning of Match Score varies between U.S. and non-U.S. addresses. The int getFieldMatchscore (unit record, const char*) field is a decimal value between 0 and 100 that reflects the similarity between the identified input data and the closest reference data match. A result of 100 indicates that no changes other than alias, casing, or diacritic changes have been made to the input data. A result of 0 indicates that there is no similarity between the input data and closest reference data match. **Note:** The Validate Address Loqate and Advanced Matching Module components both use the MatchScore field. The MatchScore field value in the output of a dataflow is determined by the last stage to modify the value before it is sent to an output stage. If you have a dataflow that contains Validate Address Loqate and Advanced Matching Module components and you want to see the MatchScore field output for each stage, use a Transformer stage to copy the MatchScore value to another field. For example, Validate Address Loqate produces an output field called MatchScore and then a Transformer stage copies the MatchScore field from Validate Address Loqate to a field called AddressMatchScore. When the matcher stage runs it populates the MatchScore field with the value from the matcher and passes through the AddressMatchScore value from Validate Address Loqate.
ProcessedBy	Which address coder processed the address: **LOQATE** The Loqate coder processed the address.
Status	Reports the success or failure of the match attempt. For multiple matches, this field is "F" for all the possible matches. **null** Success **F** Failure
Status.Code	Reason for failure, if there is one.

columnName	Description / Valid Values
	• UnableToValidate
Status.Description	Description of the problem, if there is one. **Address Not Found** This value will appear if Status.Code=UnableToValidate.

Field-Level Result Indicators

Field-level result indicators describe how ValidateAddressLoqate handled each address element. Field-level result indicators are returned in the qualifier "Result". For example, the field-level result indicator for HouseNumber is contained in **HouseNumber.Result**.

To enable field-level result indicators, specify **OutputFieldLevelReturnCodes=Y**.

The following table lists the field-level result indicators. If a particular field does not apply to an address, the result indicator may be blank.

Table 281: Field-Level Result Indicators

columnName	Valid Values
ApartmentLabel.Result	**A** Appended. The field was added to a blank input field. U.S. and Canadian addresses only. **C** Corrected. U.S. and Canadian addresses only. **D** Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About Additional Input Data** on page 884. **F** Formatted. The spacing and/or punctuation was changed to conform to postal standards. **P** Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only. **R** The apartment label is required but is missing from the input address. U.S. addresses only. **S** Standardized. This option includes any standard abbreviations. **U** Unmatched. Does not apply to Canadian addresses. **V** Validated. The data was confirmed correct and remained unchanged from input.

columnName	Valid Values	
ApartmentNumber.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. addresses that are an EWS match will have a value of P. U.S. and Canadian addresses only.
	R	The apartment number is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
City.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	F	Hyphens missing or punctuation errors. Canadian addresses only.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output.
	R	The city is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.

columnName	Valid Values	
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
Country.Result	These result codes do not apply to U.S. or Canadian addresses.	
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
FirmName.Result	**C**	Corrected. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only.
	U	Unmatched. U.S. and Canadian addresses only.
	V	Validated. The data was confirmed correct and remained unchanged from input. U.S. addresses only.
HouseNumber.Result	**A**	Appended. The field was added to a blank input field. Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	O	Out of range. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.

columnName	Valid Values	
	R	The house number is required but is missing from the input address. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. or Canadian addresses.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
LeadingDirectional.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. Non-blank input was corrected to a non-blank value. U.S. addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input. Does not apply to Canadian addresses.
POBox.Result	**A**	Appended. The field was added to a blank input field. Canadian addresses only.
	C	Corrected. Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About Additional Input Data** on page 884.

columnName	Valid Values	
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple matches. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	R	The P.O. Box number is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
PostalCode.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Does not apply to U.S. addresses.
	R	The postal code is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. or Canadian addresses.
	U	Unmatched. For example, if the street name does not match the postal code, both StreetName.Result and PostalCode.Result will contain U.
	V	Validated. The data was confirmed correct and remained unchanged from input.

columnName	Valid Values	
PostalCode.Type	**P**	The ZIP Code™ contains only PO Box addresses. U.S. addresses only.
	U	The ZIP Code™ is a unique ZIP Code™ assigned to a specific company or location. U.S. addresses only.
	M	The ZIP Code™ is for military addresses. U.S. addresses only.
	null	The ZIP Code™ is a standard ZIP Code™.
RRHC.Type	These result codes apply to U.S. addresses only.	
	HC	The address is a Highway Contract address.
	RR	The address is a Rural Route address.
StateProvince.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. addresses only.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. Does not apply to U.S. or Canadian addresses.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. U.S. and Canadian addresses only.
	R	The state is required but is missing from the input address. U.S. addresses only.
	S	Standardized. This option includes any standard abbreviations. Does not apply to U.S. addresses.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
Street.Result	These result codes apply to international addresses only.	
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field.

columnName	Valid Values	
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output.
	R	Street corrected. House number is out of range. Applies to UK and Japanese records only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
StreetName.Result	**A**	Appended. The field was added to a blank input field. Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Does not apply to U.S. addresses.
	S	Standardized. This option includes any standard abbreviations. U.S. and Canadian addresses only.
	U	Unmatched.
	V	Validated. The data was confirmed correct and remained unchanged from input.
StreetSuffix.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.

columnName	Valid Values	
	D	Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched. Does not apply to U.S. addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.
TrailingDirectional.Result	**A**	Appended. The field was added to a blank input field. U.S. and Canadian addresses only.
	C	Corrected. U.S. and Canadian addresses only.
	D	Dropped. The field provided on input was removed. U.S. and Canadian addresses only. For more information, see **About Additional Input Data** on page 884.
	F	Formatted. The spacing and/or punctuation was changed to conform to postal standards. Does not apply to U.S. or Canadian addresses.
	M	Multiple. The input address matched multiple records in the postal database, and each matching record has a different value in this field. U.S. addresses only.
	P	Pass-through. The data was not used in the validation process, but it was preserved in the output. Canadian addresses only.
	S	Standardized. This option includes any standard abbreviations.
	U	Unmatched. Does not apply to Canadian addresses.
	V	Validated. The data was confirmed correct and remained unchanged from input.

About Additional Input Data

ValidateAddressLoqate ignores some input data during the address standardization process. This extraneous data (sometimes referred to as "dropped data") is returned in the AdditionalInputData column. Some examples of dropped data include:

- Delivery instructions (for example, "Leave at back door")
- Phone numbers (for example, "555-135-8792")
- Attention lines (for example, "Attn: John Smith")

Data such as this is generally not embedded in an address. If it is embedded, ValidateAddressLoqate can usually identify this extraneous data and return it in the AdditionalInputData column.

Note: ValidateAddressLoqate does not return dropped data from split indicia addresses. A split indicia address is one where a primary address is split between multiple address lines. For example, if the primary address is "1 Green River Valley Rd" then the following would be a split indicia version of this address: 1 Green River Valley Rd 01230

If there is more than one piece of dropped data in an address, each piece of data is separated by a semicolon and a space ("; ") for U.S. addresses and a space for addresses outside the U.S. The order of dropped data in AdditionalInputData is:

1. Care of, mail stop (U.S. addresses only)
2. Other extraneous data found on address lines
3. Entire unused data lines

For example, if this is the input address:

123 Main St C/O John Smith
Apt 5 Drop at back dock
jsmith@something.com
555-123-4567
05674

Then AdditionalInputData would contain:

C/O John Smith; Apt 5 Drop At Back Dock; 555-123-4567; Jsmith@g1.Com; 555-123-4567

ValidateAddressLoqate can handle the following types of extraneous data:

- **Care Of Data** on page 884
- **Extraneous Data on Its Own Address Line** on page 885
- **Extraneous Data Within an Address Line** on page 885

Care Of Data

For U.S. addresses only, "care of" data is returned in AdditionalInputData. The following addresses contain examples of "care of" data:

123 Main St C/O John Smith
Apt 5
05674

123 Main St
Apt 5 ATTN John Smith
05674

123 Main St Apt 5
MailStop 2
05674

Extraneous Data on Its Own Address Line

ValidateAddressLoqate returns extraneous data on its own address line for U.S. and Canadian addresses.

For U.S. addresses, ValidateAddressLoqate uses the first two non-blank address lines to perform address standardization, unless either the firm name extraction or urbanization code extraction options are enabled (see **Address Line Processing for U.S. Addresses** on page 862 for more information). Data on other address lines is returned in AdditionalInputData. In the following address, "John Smith" would be returned in AdditionalInputData because it is in the third non-blank address line and ValidateAddressLoqate only uses the first two non-blank address lines for U.S. addresses.

123 Main St
Apt 5
John Smith
05674

If one of either of the first two non-blank address lines contains extraneous data, that data is returned in AdditionalInputData. For example, in the following addresses "John Smith" would be returned in AdditionalAddressData.

123 Main St
John Smith
05674

John Smith
123 Main St
05674

In the following address both "John Smith" and "Apt 5" would both be returned in AdditionalInputData. "John Smith" would be returned because it is extraneous data in one of the first two address lines and "Apt 5" would be returned because U.S. address data must be in the first two non-blank address lines.

John Smith
123 Main St
Apt 5
05674

Extraneous Data Within an Address Line

Extraneous data that is within an address line is returned in AdditionalInputData. For example, in the following addresses "John Smith" would be returned in AdditionalInputData.

123 Main St John Smith
05674

123 Main St Apt 5 John Smith
05674

123 Main St John Smith
Apt 5
05674

123 Main St
Apt 5 John Smith
05674

For U.S. addresses, only extraneous data at the end of the address line is returned in AdditionalInputData. Extraneous data that is not at the end of an address line is not returned for U.S. addresses. For example, in the following addresses "John Smith" is not returned.

John Smith 123 Main St
05674

123 Main John Smith St
05674

The AdditionalInputData column will sometimes contain the original street name or suffix if the street name was changed to obtain a match and the street name or suffix was at the end of a line. For example this address:

Pitney Bowes Software
4200 Parlament
Lanham MD

ValidateAddressLoqate would correct the spelling of the street name and add the suffix, returning "4200 Parliament Pl" as the corrected street address and "Parlament" in AdditionalInputData.

Encountering False Positives

What is a False-Positive?

To prevent the generation of address lists, the DPV and LACS^{Link} databases include false-positive records. False-positive records are artificially manufactured addresses that reside in a false-positive table. For each negative response that occurs in a DPV or LACS^{Link} query, a query is made to the false-positive table. A match to this table (called a false-positive match) disables your DPV or LACS^{Link} key. In batch processing the job that contains the violation will complete successfully but you will not be able to run any subsequent jobs that use DPV or LACS^{Link} until you report the violation and obtain a key to reactivate DPV or LACS^{Link}.

Note: The term "seed record violation" is also used to refer to encountering false positive records. The two terms mean the same thing.

Reporting DPV False-Positive Violations

Spectrum™ Technology Platform indicates a false-positive match via messages in the server log.

Client/server calls throw an exception if a false-positive match occurs. When a DPV fals positive record violation occurs, the server log will say:

```
WARN [Log] Seed record violation for S<ZIP, ZIP+4, Address, Unit> ERROR [Log]
 Feature Disabled: DPU: DPV Seed Record Violation. Seed Code: S<Address, ZIP,
 ZIP+4, Unit>
```

Note: If a DPV false positive record is found, the process() method (COM, C++, Java, and .NET) will throw an exception that the feature DPU has been disabled. In C, the processMessage() function will return a non-zero value.

You can report the violation and obtain a restart key by following these steps.

1. In your browser, go to http://<yourserver>:<port>/<product code>/dpv.jsp. For example, http://localhost:8080/unc/dpv.jsp for the Universal Addressing Module and http://localhost:8080/geostan/dpv.jsp for the Enterprise Geocoding Module.
2. Enter the mailer's information into each field. The number in parentheses after each field name indicates the maximum length of the field.
3. Click **Submit** when you're done. A **File Download** dialog will appear.
4. Click **Save** to save the file to your computer. A **Save As** dialog will appear.
5. Specify a file name and location on your local hard drive (for example `c:\DPVSeedFile.txt`) and click **Save**.
6. Go to **www.g1.com/support** and log in.
7. Click **DPV & LACS**[Link] **False Positive**.
8. Follow the on-screen instructions to attach your seed file and obtain a restart key.

DPV False Positive Header File Layout

The USPS® has determined the required layout of the DPV false-positive header file, which is currently defined as a fixed-length file containing two or more 180-byte records. The first record must always be the header record, whose layout is shown below.

Table 282: DPV False-Positive Header Record Layout

Position	Length	Description	Format
1-40	40	Mailer's company name	Alphanumeric
41-98	58	Mailer's address line	Alphanumeric
99-126	28	Mailer's city name	Alphanumeric
127-128	2	Mailer's state abbreviation	Alphabetic
129-137	9	Mailer's 9-digit ZIP Code	Numeric
138-146	9	Total Records Processed	Numeric

Position	Length	Description	Format
147-155	9	Total Records DPV Matched	Numeric
156-164	9	Percent Match Rate to DSF	Numeric
165-173	9	Percent Match Rate to ZIP + 4®	Numeric
174-178	5	Number of ZIP Codes on file	Numeric
179-180	2	Number of False-Positives	Numeric

The trailer record contains information regarding the DPV false-positive match. There must be one trailer record added to the false-positive file for every DPV false-positive match. The layout is shown below.

Table 283: DPV False-Positive Trailer Record Layout

Position	Length	Description	Format
1-2	2	Street predirectional	Alphanumeric
3-30	28	Street name	Alphanumeric
31-34	4	Street suffix abbreviation	Alphanumeric
35-36	2	Street postdirectional	Alphanumeric
37-46	10	Address primary number	Alphanumeric
47-50	4	Address secondary abbreviation	Alphanumeric
51-58	8	Address secondary number	Numeric
59-63	5	Matched ZIP Code	Numeric
64-67	4	Matched ZIP + 4®	Numeric
68-180	113	Filler	Spaces

Reporting LACS/Link False-Positive Violations

Spectrum™ Technology Platform indicates a false-positive match via messages in the server log. Batch jobs will fail if a false-positive match occurs and client/server calls will throw an exception.

Note: The term "seed record violation" is also used to refer to encountering false positive records. The two terms mean the same thing.

When a false positive record is encountered, the server log will say:

```
2005-05-06 17:05:38,978 WARN [com.g1.component.ValidateAddress] Seed record
 violation for RR 2 28562  31373
2005-05-06 17:05:38,978 ERROR [com.g1.component.ValidateAddress] Feature
Disabled: LLU: LACS Seed Record Violation. Seed Code: 28562  31373
2005-05-06 17:05:38,978 ERROR [com.g1.dcg.gateway.Gateway] Gateway exception:
 com.g1.dcg.stage.StageException: com.g1.dcg.component.ComponentException:
Feature Disabled: LLU
2005-05-06 17:06:30,291 ERROR [com.g1.dcg.server.license.policy.Policy]
Feature LACSLink Real- time is disabled.
```

Note: If a LACS^{Link} false positive record is found, the process() method (COM, C++, Java, and .NET) will throw an exception that the feature LLU has been disabled. In C, the processMessage() function will return a non-zero value.

1. In your browser, go to http://<ServerName>:<port>/<product code>/lacslink.jsp. For example, http://localhost:8080/unc/lacslink.jsp for the Universal Addressing Module and http://localhost:8080/geostan/lacslink.jsp for the Enterprise Geocoding Module.
2. Enter the mailer's information into each field. The number in parentheses after the field name indicates the maximum length of the field. Click **Submit** when you're done. A **File Download** dialog will appear.
3. Click **Save** to save the file to your computer. A **Save As** dialog will appear.
4. Specify a file name and location on your local hard drive (for example `c:\lacslink.txt`) and click **Save**.
5. Go to **www.g1.com/support** and log in.
6. Click **DPV & LACS^{Link} False Positive**.
7. Follow the on-screen instructions to attach your seed file and obtain a restart key.

ValidateAddress Confidence Algorithm

Introduction to the Validate Address Confidence Algorithm

ValidateAddress computes a confidence score for each validated address. This score describes how likely it is that the validated address is correct. Confidence code values range from 0 to 100, with a zero confidence level indicating no confidence and 100 indicating a very high level of confidence that the match results are correct. Confidence codes are calculated based on an algorithm that takes into account the match results for individual output fields. The output fields involved in this calculation include:

- Country
- City

- State
- PostalCode
- StreetName
- HouseNumber
- LeadingDirectional
- TrailingDirectional
- StreetSuffix
- ApartmentNumber

Each field has its own Weight in the algorithm. Additionally, for each field the match result could be labeled as Success, Failure, or Changed. ("Changed" refers to cases where the contents of the field have been corrected in order to get a match.) The match result—Success, Failure, or Changed—determines what the Factor is for that field. Thus, the calculation for the confidence code is a product of Weight by Factor as follows:

```
Confidence = (Weight * Factor) for City
  + (Weight * Factor) for Country
  + (Weight * Factor) for State
  + (Weight * Factor) for PostalCode
  + (Weight * Factor) for StreetName
  + (Weight * Factor) for HouseNumber
  + (Weight * Factor) for Directionals
  + (Weight * Factor) for Street Suffix
  + (Weight * Factor) for ApartmentNumber
```

Confidence Algorithm for U.S. and Canadian Addresses

The following table details the scoring and logic behind the ValidateAddress confidence algorithm for U.S. and Canadian addresses.

Table 284: Confidence Algorithm for U.S. and Canadian Addresses

Field	Weight/Match Score	Factor if Changed[3]	Factor If Filled[4]
Country	10	100%	0%
City	10	50%	75%
State	15	50%	75%
PostalCode	15	25%	25%
StreetName	15	50%	75%
HouseNumber	15	50%	75%

Field	Weight/Match Score	Factor if Changed[3]	Factor If Filled[4]
Directionals	10	50%	75%
StreetSuffix	5	50%	75%
ApartmentNumber	5	50%	75%

Confidence Algorithm for International Addresses

There are two confidence algorithms for addresses outside the U.S. and Canada, one for addresses in countries that use postal codes and one for addresses in countries that do not use postal codes.

The following table details the confidence algorithm for non-U.S. and non-Canadian addresses from countries that use postal codes.

Table 285: Confidence Algorithm for Countries With Postal Codes

Field	Weight/Match Score	Factor if Changed[5]	Factor If Filled[6]	Factor if Postal Data Unavailable
Country	11.1111111111111	100%	0%	0%
City	11.1111111111111	50%	75%[7]	0%
State	16.6666666666667	100%	100	80%
PostalCode	16.6666666666667	100%	100%	80%
StreetName	16.6666666666667	50%	75%	50%
HouseNumber	16.6666666666667	50%	75%	50%
Directionals	0	50%	75%	0%

[7] If the country is a Category C country, this value is 50%. For a description of country categories, see **Universal Addressing Databases** on page 722.

Field	Weight/Match Score	Factor if Changed[5]	Factor If Filled[6]	Factor if Postal Data Unavailable
StreetSuffix	5.55555555555556	50%	75%	50%
ApartmentNumber	5.55555555555556	50%	75%	50%

The following table details confidence algorithm for countries that do not use postal codes.

Table 286: Confidence Algorithm for Countries Without Postal Codes

Field	Weight/Match Score	Factor if Changed[8]	Factor If Filled[9]	Factor if Postal Data Unavailable
Country	13.3333333333333	100%	0%	0%
City	13.3333333333333	50%	75%[10]	0%
State	20	100%	100	80%
StreetName	20	50%	75%	50%
HouseNumber	20	50%	75%	50%
Directionals	0	50%	75%	0%
StreetSuffix	6.66666666666667	50%	75%	50%
ApartmentNumber	6.66666666666667	50%	75%	50%

The following table lists countries without postal codes.

[10] If the country is a Category C country, this value is 50%. For a description of country categories, see **Universal Addressing Databases** on page 722.

Table 287: Countries Without Postal Codes

Afghanistan	Albania	Angola
Anguilla	Bahamas	Barbados
Belize	Benin	Bhutan
Botswana	Burkina Faso	Burundi
Cameroon	Cayman Islands	Central African Rep.
Chad	Cocos Islands	Columbia
Comoros	Congo (Dem. Rep.)	Congo (Rep.)
Cote d'Ivoire	Korea (North)	Djibouti
Dominica	Equatorial Guinea	Eritrea
Fiji	Gabon	Gambia
Ghana	Grenada	Guyana
Ireland	Jamaica	Kiribati
Libya	Malawi	Mali
Mauritania	Namibia	Nauru
Palaos	Panama	Peru
Qatar	Rwanda	Saint Lucia
Saint Vincent & Grenadines	Samoa	Sao Tome & Principe
Seychelles	Sierra Leone	Suriname
Tanzania	Timor	Togo
Tonga	Trinidad & Tobago	Tuvalu
Uganda	United Arab Emirates	Vanuatu

Yemen	Zimbabwe

Appendices

In this section:

- Message XML Schema 897
- Module Matrix 903
- Country ISO Codes and Module Support 907

A

Message XML Schema

In this section:

- What Is the Message XML Schema? 898
- Message XML Schema Examples 899

What Is the Message XML Schema?

Message XML contains propertysets and a dataset. The propertysets includes context properties, such as account ID, account password, service name; option properties, which are the service-specific runtime options; and error properties, which are the error message(s) from server. The dataset includes all column names and the corresponding field values.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="column">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="field">
    <xs:complexType>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="propertysets">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="propertyset" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="dataset">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="columns">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="column" maxOccurs="unbounded"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="records">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="record" maxOccurs="unbounded"/>
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element ref="field" maxOccurs="unbounded"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="property">
    <xs:complexType>
      <xs:attribute name="key" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="propertyset">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="context"/>
            <xs:enumeration value="option"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Message XML Schema Examples

These examples illustrate how the XML is used.

Sending Message to the Server

The example below shows the XML the client would send to the server.

```
POST /dcg/gateway HTTP/1.1
Host: asp.g1.com
Connection: Close
Content-Length: 1153
Content-Type: text/xml
x-g1-connection-version: 1.0
x-g1-message-serialization: g1-xml
x-g1-message-encoding: clear
<html xmlns="http://www.w3.org/1999/xhtml">
        <propertysets>
                <propertyset name="context">
                        <property key="account.id" value="qa"/>
                        <property key="account.password" value="wendy"/>
                     <property key="service.name" value="Validate Address"/>

                </propertyset>
                <propertyset name="option">
                        <property key="OutputCasing" value="M"/>
                        <property key="OutputFormattedOnFail" value="Y"/>
                        <property key="OutputMultinationalCharacters"
value="Y"/>
                      <property key="OutputPostalCodeSeparator" value="Y"/>
```

```
                        <property key="OutputRecordType" value="A"/>
                </propertyset>
        </propertysets>
        <dataset>
                <columns>
                        <column name="InputKeyValue"/>
                        <column name="FirmName"/>
                        <column name="AddressLine1"/>
                        <column name="AddressLine2"/>
                        <column name="City"/>
                        <column name="StateProvince"/>
                        <column name="PostalCode"/>
                </columns>
                <records>
                        <record>
                                <field value="2"/>
                                <field value="Gary W Crawford"/>
                                <field value="510 S Coit St"/>
                                <field value=""/>
                                <field value="Florence"/>
                                <field value="SC"/>
                                <field value="29501-5221"/>
                        </record>
                </records>
        </dataset>
</html>
```

Receiving Message from the Server

The example below shows the XML the client receives from the server.

```
HTTP/1.1 200 OK
Date: Thu, 13 Jan 2005 18:40:55 GMT
Server: Jetty/4.2.15 (Windows 2000/5.0 x86 java/1.3.1_09)
Connection: close
Content-Type: text/xml
x-g1-message-encoding: clear
x-g1-message-serialization: g1-xml
Content-Length: 2384
<?xml version="1.0" encoding="UTF-8"?>
<message>
 <propertysets>
 <propertyset name="context">
 <property key="context.timestamp" value="1105641655192"/>
 </propertyset><
 /propertysets>
 <dataset>
 <columns>
 <column name="Status"/>
 <column name="Status.Code"/>
 <column name="Status.Description"/>
 <column name="Confidence"/>
 <column name="ProcessedBy"/>
 <column name="RecordType"/>
 <column name="MultipleMatches"/>
 <column name="CouldNotValidate"/>
 <column name="CountryLevel"/>
 <column name="AddressFormat"/>
 <column name="FirmName"/>
 <column name="USUrbanName"/>
```

```
<column name="AddressLine1"/>
<column name="AddressLine2"/>
<column name="AddressLine3"/>
<column name="AddressLine4"/>
<column name="City"/>
<column name="StateProvince"/>
<column name="PostalCode"/>
<column name="Country"/>
<column name="AdditionalInputData"/>
<column name="HouseNumber.Result"/>
<column name="LeadingDirectional.Result"/>
<column name="StreetName.Result"/>
<column name="StreetName.AliasType"/>
<column name="StreetSuffix.Result"/>
<column name="TrailingDirectional.Result"/>
<column name="ApartmentLabel.Result"/>
<column name="ApartmentNumber.Result"/>
<column name="ApartmentNumber.Input.Result"/>
<column name="RRHC.Result"/>
<column name="RRHC.Type"/>
<column name="POBoxNumber.Result"/>
<column name="PrivateMailbox.Result"/>
<column name="City.Result"/>
<column name="StateProvince.Result"/>
<column name="PostalCode.Result"/>
<column name="PostalCode.Source"/>
<column name="Country.Result"/>
<column name="FirmName.Result"/>
<column name="USUrbanName.Result"/>
<column name="InputKeyValue"/>
</columns>
<records>
<record>
<field value="S"/>
<field value=""/>
<field value=""/>
<field value="100"/>
<field value="C1P"/>
<field value="Normal"/>
<field value=""/>
<field value=""/>
<field value="A"/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value="510 S Coit St"/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value="Florence"/>
<field value="SC"/>
<field value="29501-5221"/>
<field value="United States Of America"/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value=""/>
<field value=""/>
```

```
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value=""/>
 <field value="V"/>
 <field value=""/>
 <field value=""/>
 <field value="P"/>
 <field value=""/>
 <field value="2"/>
 </record>
 </records>
</dataset>
</message>
```

B

Module Matrix

In this section:

- Modules, Components, and Databases 904

Modules, Components, and Databases

Table 288: Modules, Components, and Databases

Module	Description	Components
Address Now Module	Provides enhanced validation and standardization for addresses outside the U.S., and other address processing.	Build Global Address Get Global Candidate Addresses Validate Global Address
Advanced Matching Module	Matches records within and/or between input files	Best Of Breed Candidate Finder Duplicate Synchronization Filter Interflow Match Intraflow Match Match Key Generator Transactional Match
Business Steward Module	Identifies exception records and provides a browser-based tool for manually reviewing exception records.	Exception Monitor Read Exceptions Write Exceptions
Country Identifier	Takes a country name or a combination of postal code and state/province and returns the two-character ISO country code, the three-character Universal Postal Union (UPU) code, and the English country name.	Country Identifier
Data Hub Module	Links and analyzes data, identifying relationships and trends.	Write to Hub Read From Hub Query Hub Graph Visualization
Data Normalization Module	Removes inconsistencies in data.	Advanced Transformer Open Parser Table Lookup

Module	Description	Components
		Transliterator
Enterprise Geocoding Module	Determines the geographic coordinates for an address. Also determines the address of a given latitude and longitude.	GeocodeAddressAUS GeocodeAddressGBR GeocodeAddressGlobal GeocodeAddressWorld GeocodeUSAddress GNAFPIDLocationSearch ReverseAPNLookup ReverseGeocodeAddressGlobal ReverseGeocodeUSLocation
Enterprise Routing Module	Obtains driving or walking directions, calculates drive time and drive distance, and identifies locations within a certain time or distance from a starting point.	GetTravelBoundary GetTravelCostMatrix GetTravelDirections
Enterprise Tax Module	Determines the tax jurisdictions that apply to a given location.	AssignGeoTAXInfo CalculateDistance
GeoConfidence Module	Determines the probability that an address or street intersection is within a given area.	Geo Confidence Surface CreatePointsConvexHull
Global Sentry	Attempts to match transactions against government-provided watch lists that contain data from different countries.	Global Sentry Global Sentry Address Check Global Sentry ID Number Check Global Sentry Name Check Global Sentry Other Data Check
Location Intelligence Module	Performs point in polygon and radial analysis against a variety of geospatial databases.	Closest Site Find Nearest Point In Polygon Query Spatial Data Read Spatial Data Spatial Calculator Spatial Union

Module	Description	Components
SAP Module	Enables Spectrum™ Technology Platform to interface with SAP Customer Relationship Management Module applications.	SAP Generate Match Key SAP Generate Match Score SAP Generate Search Key SAP Generate Search Key Constant SAP Generate Search Key Metaphone SAP Generate Search Key Substring SAP Validate Address With Candidates
Siebel Module	Enables Spectrum™ Technology Platform to interface Siebel applications	Siebel Generate Match Key Siebel Generate Match Score Siebel Generate Search Key Siebel Business Name Standardization Siebel Standardize Name Siebel Geocode US Address With Candidates Siebel Geocode US Address With No Candidates Siebel Get Global Candidate Addresses Siebel Validate Address With Candidates Siebel Validate Address With No Candidates
Universal Addressing Module	Standardizes and validates addresses according to the postal authority's standards	Get Candidate Addresses Get City State Province Get Postal Codes Validate Address Validate Address AUS Validate Address Global
Universal Name Module	Parses personal names, company names, addresses, and many other terms and abbreviations.	Name Parser Name Variant Finder

C

Country ISO Codes and Module Support

In this section:

- Country ISO Codes and Module Support 908

Country ISO Codes and Module Support

The following table lists the ISO codes for each country as well as the modules that support addressing, geocoding, and routing for each country.

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Afghanistan	AF	AFG	Address Now Module Universal Addressing Module
Aland Islands	AX	ALA	Address Now Module Universal Addressing Module
Albania	AL	ALB	Address Now Module Universal Addressing Module
Algeria	DZ	DZA	Address Now Module Universal Addressing Module
American Samoa	AS	ASM	Address Now Module Universal Addressing Module
Andorra	AD	AND	Address Now Module Enterprise Geocoding Module[11] Universal Addressing Module
Angola	AO	AGO	Address Now Module Universal Addressing Module
Anguilla	AI	AIA	Address Now Module Universal Addressing Module
Antarctica	AQ	ATA	Address Now Module Universal Addressing Module

[11] Andorra is covered by the Spain geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Antigua And Barbuda	AG	ATG	Address Now Module Universal Addressing Module
Argentina	AR	ARG	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Armenia	AM	ARM	Address Now Module Universal Addressing Module
Aruba	AW	ABW	Address Now Module Universal Addressing Module
Australia	AU	AUS	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Austria	AT	AUT	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Azerbaijan	AZ	AZE	Address Now Module Universal Addressing Module
Bahamas	BS	BHS	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Bahrain	BH	BHR	Address Now Module Universal Addressing Module
Bangladesh	BD	BGD	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Barbados	BB	BRB	Address Now Module Universal Addressing Module
Belarus	BY	BLR	Address Now Module Universal Addressing Module
Belgium	BE	BEL	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Belize	BZ	BLZ	Address Now Module Universal Addressing Module
Benin	BJ	BEN	Address Now Module Universal Addressing Module
Bermuda	BM	BMU	Address Now Module Universal Addressing Module
Bhutan	BT	BTN	Address Now Module Universal Addressing Module
Bolivia, Plurinational State Of	BO	BOL	Address Now Module Universal Addressing Module
Bonaire, Saint Eustatius And Saba	BQ	BES	Address Now Module Universal Addressing Module
Bosnia And Herzegovina	BA	BIH	Address Now Module Universal Addressing Module
Botswana	BW	BWA	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Bouvet Island	BV	BVT	Address Now Module Universal Addressing Module
Brazil	BR	BRA	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
British Indian Ocean Territory	IO	IOT	Address Now Module Universal Addressing Module
Brunei Darussalam	BN	BRN	Address Now Module Universal Addressing Module
Bulgaria	BG	BGR	Address Now Module Universal Addressing Module
Burkina Faso	BF	BFA	Address Now Module Universal Addressing Module
Burundi	BI	BDI	Address Now Module Universal Addressing Module
Cambodia	KH	KHM	Address Now Module Universal Addressing Module
Cameroon	CM	CMR	Address Now Module Universal Addressing Module
Canada	CA	CAN	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Cape Verde	CV	CPV	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Cayman Islands	KY	CYM	Address Now Module Universal Addressing Module
Central African Republic	CF	CAF	Address Now Module Universal Addressing Module
Chad	TD	TCD	Address Now Module Universal Addressing Module
Chile	CL	CHL	Address Now Module Enterprise Geocoding Module Universal Addressing Module
China	CN	CHN	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Christmas Island	CX	CXR	Address Now Module Universal Addressing Module
Cocos (Keeling) Islands	CC	CCK	Address Now Module Universal Addressing Module
Colombia	CO	COL	Address Now Module Universal Addressing Module
Comoros	KM	COM	Address Now Module Universal Addressing Module
Congo	CG	COG	Address Now Module Universal Addressing Module
Congo, The Democratic Republic Of The	CD	COD	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Cook Islands	CK	COK	Address Now Module Universal Addressing Module
Costa Rica	CR	CRI	Address Now Module Universal Addressing Module
Côte d'Ivoire	CI	CIV	Address Now Module Universal Addressing Module
Croatia	HR	HRV	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Cuba	CU	CUB	Address Now Module Universal Addressing Module
Curacao	CW	CUW	Address Now Module Universal Addressing Module
Cyprus	CY	CYP	Address Now Module Universal Addressing Module
Czech Republic	CZ	CZE	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Denmark	DK	DNK	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Djibouti	DJ	DJI	Address Now Module Universal Addressing Module
Dominica	DM	DMA	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Dominican Republic	DO	DOM	Address Now Module Universal Addressing Module
Ecuador	EC	ECU	Address Now Module Universal Addressing Module
Egypt	EG	EGY	Address Now Module Universal Addressing Module
El Salvador	SV	SLV	Address Now Module Universal Addressing Module
Equatorial Guinea	GQ	GNQ	Address Now Module Universal Addressing Module
Eritrea	ER	ERI	Address Now Module Universal Addressing Module
Estonia	EE	EST	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Ethiopia	ET	ETH	Address Now Module Universal Addressing Module
Falkland Islands (Malvinas)	FK	FLK	Address Now Module Universal Addressing Module
Faroe Islands	FO	FRO	Address Now Module Universal Addressing Module
Fiji	FJ	FJI	Address Now Module Universal Addressing Module
Finland	FI	FIN	Address Now Module Enterprise Geocoding Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
			Location Intelligence Module Universal Addressing Module
France	FR	FRA	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
French Guiana	GF	GUF	Address Now Module Universal Addressing Module
French Polynesia	PF	PYF	Address Now Module Universal Addressing Module
French Southern Territories	TF	ATF	Address Now Module Universal Addressing Module
Gabon	GA	GAB	Address Now Module Universal Addressing Module
Gambia	GM	GMB	Address Now Module Universal Addressing Module
Georgia	GE	GEO	Address Now Module Universal Addressing Module
Germany	DE	DEU	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Ghana	GH	GHA	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Gibraltar	GI	GIB	Address Now Module Enterprise Geocoding Module [12] Universal Addressing Module
Greece	GR	GRC	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Greenland	GL	GRL	Address Now Module Universal Addressing Module
Grenada	GD	GRD	Address Now Module Universal Addressing Module
Guadeloupe	GP	GLP	Address Now Module Universal Addressing Module
Guam	GU	GUM	Address Now Module Universal Addressing Module
Guatemala	GT	GTM	Address Now Module Universal Addressing Module
Guernsey	GG	GGY	Address Now Module Universal Addressing Module
Guinea	GN	GIN	Address Now Module Universal Addressing Module
Guinea-Bissau	GW	GNB	Address Now Module Universal Addressing Module
Guyana	GY	GUY	Address Now Module Universal Addressing Module

[12] Gibraltar is covered by the Spain geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Haiti	HT	HTI	Address Now Module Universal Addressing Module
Heard Island and McDonald Islands	HM	HMD	Address Now Module Universal Addressing Module
Holy See (Vatican City State)	VA	VAT	Address Now Module Enterprise Geocoding Module [13] Universal Addressing Module
Honduras	HN	HND	Address Now Module Universal Addressing Module
Hong Kong	HK	HKG	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Hungary	HU	HUN	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Iceland	IS	ISL	Address Now Module Universal Addressing Module
India	IN	IND	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Indonesia	ID	IDN	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Iran, Islamic Republic Of	IR	IRN	Address Now Module Universal Addressing Module

[13] The Vatican is covered by the Italy geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Iraq	IQ	IRQ	Address Now Module Universal Addressing Module
Ireland	IE	IRL	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Isle Of Man	IM	IMN	Address Now Module Universal Addressing Module
Israel	IL	ISR	Address Now Module Universal Addressing Module
Italy	IT	ITA	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Jamaica	JM	JAM	Address Now Module Universal Addressing Module
Japan	JP	JPN	Address Now Module Enterprise Geocoding Module Enterprise Routing Module Universal Addressing Module
Jersey	JE	JEY	Address Now Module Universal Addressing Module
Jordan	JO	JOR	Address Now Module Universal Addressing Module
Kazakhstan	KZ	KAZ	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Kenya	KE	KEN	Address Now Module Universal Addressing Module
Kiribati	KI	KIR	Address Now Module Universal Addressing Module
Korea, Democratic People's Republic Of	KP	PRK	Address Now Module Universal Addressing Module
Korea, Republic Of	KR	KOR	Address Now Module Universal Addressing Module
Kosovo	KS	KOS	Address Now Module Universal Addressing Module
Kuwait	KW	KWT	Address Now Module Universal Addressing Module
Kyrgyzstan	KG	KGZ	Address Now Module Universal Addressing Module
Lao People's Democratic Republic	LA	LAO	Address Now Module Universal Addressing Module
Latvia	LV	LVA	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Lebanon	LB	LBN	Address Now Module Universal Addressing Module
Lesotho	LS	LSO	Address Now Module Universal Addressing Module
Liberia	LR	LBR	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Libyan Arab Jamahiriya	LY	LBY	Address Now Module Universal Addressing Module
Liechtenstein	LI	LIE	Address Now Module Enterprise Geocoding Module [14] Location Intelligence Module Universal Addressing Module
Lithuania	LT	LTU	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Luxembourg	LU	LUX	Address Now Module Enterprise Geocoding Module [15] Location Intelligence Module Universal Addressing Module
Macao	MO	MAC	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Macedonia, Former Yugoslav Republic Of	MK	MKD	Address Now Module Universal Addressing Module
Madagascar	MG	MDG	Address Now Module Universal Addressing Module
Malawi	MW	MWI	Address Now Module Universal Addressing Module
Malaysia	MY	MYS	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module

[14] Liechtenstein is covered by the Swizerland geocoder
[15] Luxembourg is covered by the Belgium geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Maldives	MV	MDV	Address Now Module Universal Addressing Module
Mali	ML	MLI	Address Now Module Universal Addressing Module
Malta	ML	MLT	Address Now Module Universal Addressing Module
Marshall Islands	MH	MHL	Address Now Module Universal Addressing Module
Martinique	MQ	MTQ	Address Now Module Universal Addressing Module
Mauritania	MR	MRT	Address Now Module Universal Addressing Module
Mauritius	MU	MUS	Address Now Module Universal Addressing Module
Mayotte	YT	MYT	Address Now Module Universal Addressing Module
Mexico	MX	MEX	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Micronesia, Federated States Of	FM	FSM	Address Now Module Universal Addressing Module
Moldova, Republic Of	MD	MDA	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Monaco	MC	MCO	Address Now Module Enterprise Geocoding Module [16] Universal Addressing Module
Mongolia	MN	MNG	Address Now Module Universal Addressing Module
Montenegro	ME	MNE	Address Now Module Universal Addressing Module
Montserrat	MS	MSR	Address Now Module Universal Addressing Module
Morocco	MA	MAR	Address Now Module Universal Addressing Module
Mozambique	MZ	MOZ	Address Now Module Universal Addressing Module
Myanmar	MM	MMR	Address Now Module Universal Addressing Module
Namibia	NA	NAM	Address Now Module Universal Addressing Module
Nauru	NR	NRU	Address Now Module Universal Addressing Module
Nepal	NP	NPL	Address Now Module Universal Addressing Module
Netherlands	NL	NLD	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module

[16] Monaco is covered by the France geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
New Caledonia	NC	NCL	Address Now Module Universal Addressing Module
New Zealand	NZ	NZL	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Nicaragua	NI	NIC	Address Now Module Universal Addressing Module
Niger	NE	NER	Address Now Module Universal Addressing Module
Nigeria	NG	NGA	Address Now Module Universal Addressing Module
Niue	NU	NIU	Address Now Module Universal Addressing Module
Norfolk Island	NF	NFK	Address Now Module Universal Addressing Module
Northern Mariana Islands	MP	MNP	Address Now Module Universal Addressing Module
Norway	NO	NOR	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Oman	OM	OMN	Address Now Module Universal Addressing Module
Pakistan	PK	PAK	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Palau	PW	PLW	Address Now Module Universal Addressing Module
Palestinian Territory, Occupied	PS	PSE	Address Now Module Universal Addressing Module
Panama	PA	PAN	Address Now Module Universal Addressing Module
Papua New Guinea	PG	PNG	Address Now Module Universal Addressing Module
Paraguay	PY	PRY	Address Now Module Universal Addressing Module
Peru	PE	PER	Address Now Module Universal Addressing Module
Philippines	PH	PHL	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Pitcairn	PN	PCN	Address Now Module Universal Addressing Module
Poland	PL	POL	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Portugal	PT	PRT	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Puerto Rico	PR	PRI	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Qatar	QA	QAT	Address Now Module Universal Addressing Module
Reunion	RE	REU	Address Now Module Universal Addressing Module
Romania	RO	ROU	Address Now Module Universal Addressing Module
Russian Federation	RU	RUS	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Rwanda	RW	RWA	Address Now Module Universal Addressing Module
Saint Barthelemy	BL	BLM	Address Now Module Universal Addressing Module
Saint Helena, Ascension & Tristan Da Cunha	SH	SHE	Address Now Module Universal Addressing Module
Saint Kitts and Nevis	KN	KNA	Address Now Module Universal Addressing Module
Saint Lucia	LC	LCA	Address Now Module Universal Addressing Module
Saint Martin (French Part)	MF	MAF	Address Now Module Universal Addressing Module
Saint Pierre and Miquelon	PM	SPM	Address Now Module Universal Addressing Module
Saint Vincent And The Grenadines	VC	VCT	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Samoa	WS	WSM	Address Now Module Universal Addressing Module
San Marino	SM	SMR	Address Now Module Enterprise Geocoding Module [17] Universal Addressing Module
Sao Tome And Principe	ST	STP	Address Now Module Universal Addressing Module
Saudi Arabia	SA	SAU	Address Now Module Universal Addressing Module
Senegal	SN	SEN	Address Now Module Universal Addressing Module
Serbia	RS	SRB	Address Now Module Universal Addressing Module
Seychelles	SC	SYC	Address Now Module Universal Addressing Module
Sierra Leone	SL	SLE	Address Now Module Universal Addressing Module
Singapore	SG	SGP	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Sint Maarten (Dutch Part)	SX	SXM	Universal Addressing Module
Slovakia	SK	SVK	Address Now Module Enterprise Geocoding Module Universal Addressing Module

[17] San Marino is covered by the Italy geocoder

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Slovenia	SI	SVN	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Solomon Islands	SB	SLB	Address Now Module Universal Addressing Module
Somalia	SO	SOM	Address Now Module Universal Addressing Module
South Africa	ZA	ZAF	Address Now Module Enterprise Geocoding Module Universal Addressing Module
South Georgia And The South Sandwich Islands	GS	SGS	Address Now Module Universal Addressing Module
South Sudan	SS	SSD	Address Now Module Universal Addressing Module
Spain	ES	ESP	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Sri Lanka	LK	LKA	Address Now Module Universal Addressing Module
Sudan	SD	SDN	Address Now Module Universal Addressing Module
Suriname	SR	SUR	Address Now Module Universal Addressing Module
Svalbard And Jan Mayen	SJ	SJM	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Swaziland	SZ	SWZ	Address Now Module Universal Addressing Module
Sweden	SE	SWE	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Switzerland	CH	CHE	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Syrian Arab Republic	SY	SYR	Address Now Module Universal Addressing Module
Taiwan, Province of China	TW	TWN	Address Now Module Universal Addressing Module
Tajikistan	TJ	TJK	Address Now Module Universal Addressing Module
Tanzania, United Republic Of	TZ	TZA	Address Now Module Universal Addressing Module
Thailand	TH	THA	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
Timor-Leste	TL	TLS	Address Now Module Universal Addressing Module
Togo	TG	TGO	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Tokelau	TK	TKL	Address Now Module Universal Addressing Module
Tonga	TO	TON	Address Now Module Universal Addressing Module
Trinidad and Tobago	TT	TTO	Address Now Module Universal Addressing Module
Tunisia	TN	TUN	Address Now Module Universal Addressing Module
Turkey	TR	TUR	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Turkmenistan	TM	TKM	Address Now Module Universal Addressing Module
Turks And Caicos Islands	TC	TCA	Address Now Module Universal Addressing Module
Tuvalu	TV	TUV	Address Now Module Universal Addressing Module
Uganda	UG	UGA	Address Now Module Universal Addressing Module
Ukraine	UA	UKR	Address Now Module Universal Addressing Module
United Arab Emirates	AE	ARE	Address Now Module Universal Addressing Module
United Kingdom	GB	GBR	Address Now Module Enterprise Geocoding Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
			Location Intelligence Module Universal Addressing Module
United States	US	USA	Address Now Module Enterprise Geocoding Module Location Intelligence Module Universal Addressing Module
United States Minor Outlying Islands	UM	UMI	Address Now Module Universal Addressing Module
Uruguay	UY	URY	Address Now Module Enterprise Geocoding Module Universal Addressing Module
Uzbekistan	UZ	UZB	Address Now Module Universal Addressing Module
Vanuatu	VU	VUT	Address Now Module Universal Addressing Module
Venezuela, Bolivarian Republic Of	VE	VEN	Address Now Module Universal Addressing Module
Viet Nam	VN	VNM	Address Now Module Universal Addressing Module
Virgin Islands, British	VG	VGB	Address Now Module Universal Addressing Module
Virgin Islands, U.S.	VI	VIR	Address Now Module Universal Addressing Module
Wallis and Futuna	WF	WLF	Address Now Module Universal Addressing Module

ISO Country Name (English)	ISO 3116-1 Alpha-2	ISO 3116-1 Alpha-3	Supported Modules
Western Sahara	EH	ESH	Address Now Module Universal Addressing Module
Yemen	YE	YEM	Address Now Module Universal Addressing Module
Zambia	ZM	ZMB	Address Now Module Universal Addressing Module
Zimbabwe	ZW	ZWE	Address Now Module Universal Addressing Module

Index

.NET API
    Constants 216
    error messages 217
    Message 223
    Server 219
    Service 222
.NET Framework API 21

A

AcceptanceLevel 279
ACR, See Address Correction Result Code
Action 245, 247, 248
    output 252
AddColumn
    C API 90
    C++ API 134
    COM 165
    Java 199
AdditionalInputData 803, 805, 869
    output 531, 559
Address
    standardized 767, 861
    when no match is found 767, 861
Address Correction Result Code 289
    ACRCode field 287
    component status 290
    validation level 290
    validation match score 291
address location codes 575
Address Now Module 904
    BuildGlobalAddresses 244
    described 244
    GetGlobalCandidateAddresses 244
    ValidateGlobalAddress 244
address validation 14
AddressFormat 811
AddressLine
    output 284, 484
AddressLine1 734, 738, 742, 748, 751, 755, 767, 803, 834, 862, 869
    input 465
    output 252, 260, 284, 323, 344, 409, 492, 517, 531, 545, 559, 669
AddressLine2 734, 738, 742, 748, 751, 755, 768, 803, 834, 862
    input 465
    output 252, 261, 284, 323, 344, 409, 492, 517, 531, 545, 559, 669
AddressLine3 742, 748, 751, 755, 768, 803, 834, 862
    input 465
    output 252, 261, 284, 670
AddressLine4 742, 748, 752, 755, 768, 804, 834, 862
    input 465
    output 252, 261, 284, 670
AddressLine5 742, 748, 768, 804
    input 465
    output 252, 261, 284
AddressLine6
    input 465
    output 252, 261, 284
AddressLine7
    output 261, 284
    outputoutput
        AddressLine7 252
AddressLine8
    output 252, 261
AddressLineResolved
    output 503
AddressLineSearchOnFail 747, 802
AddressPointInterpolation
    input option 473
AddressPreference
    input 302
AddressRecord
    Result Codes 815
AddRow
    C API 94

AddRow *(continued)*
C++ API 138
COM 168
Java 201
Advanced Matching Module 904
Alternate
output 506, 537, 566
AlternativeContext 247
input 248
output 252
AlternativeIndex 247, 249
output 252
Alternatives.InContext
output 247
Alternatives.InContext.Count
output 252
Alternatives.OutContext
output 252
Alternatives.OutContext.Count
output 253
AlternativesInContext
output 252
AlternativesOutContext
output 247
AlwaysFindCandidates
input option 491
Ant 42
ApartmentLabel 805, 807, 870, 872
output 253, 261, 284, 323, 344, 409, 499, 517, 533, 545, 562
Result Codes 816, 876
ApartmentLabel2 805
output 499, 534, 562
ApartmentNumber 805, 807, 870, 872
output 253, 261, 284, 323, 344, 409, 499, 517, 534, 545, 562
Result Codes 816, 877
ApartmentNumber2 805, 870
output 499, 534, 562
API 21
client timeouts 30
components 26
introduction 26
overview 26
sample applications 29
Secure Socket Layer 29
steps for using 30
APN 526
See also Assessor's Parcel Number
output 485, 531, 558
See also Assessor's Parcel Number
architecture 18
Assessor's Parcel Number 526
Assign GeoTAX Info 651
Assign GeoTAX Info Summary Report 690
AUS.GNAF_CONFIDENCE
output 328, 421, 522
AUS.GNAF_EIGHT_DIGIT_LATITUDE
output 328, 421, 523
AUS.GNAF_EIGHT_DIGIT_LONGITUDE
output 329, 421, 523
AUS.GNAF_GEOCODE_LEVEL
output 329, 421, 523
AUS.GNAF_PID
output 329, 422, 524
AUS.GNAF_RELIABILITY
output 330, 422, 524
AUS.LEVEL_NUMBER
output 331, 423, 525
AUS.LEVEL_TYPE
output 331, 424, 525
AUS.MESH_BLOCK_ID
output 332, 424, 526
Auto Complete Loqate
inputs
AddressLine1 734
AddressLine2 734
City 734
Country 735
FirmName 735
PostalCode 735
StateProvince 735
AutoCompleteLoqate
Inputs
MaximumResults 735
output
AddressLine1 738
AddressLine2 738
City 738
Country 738
FirmName 738
HouseNumber 738
PostalCode 739
PostalCode.AddOn 739
StateProvince 739
Status.Code 739
Status.Description 739
outputs
ProcessedBy 739
auxiliary file 568
auxiliary files 298
AuxiliaryData
output 484
AuxiliaryData.AuxiliaryFile
output 667
AuxiliaryData.StateFile
output 667

B

BEA WebLogic 8.1.1
  adding Client API classes and libraries 45
  building deployment descriptors 44, 45
  deploying the Client API EJB application 45
  enterprise archive (EAR) file 44
  packaging the Client API EJB 43
bean implementation class 35
BlockLeft
  output 507, 538, 567
BlockLine
  output 484
BlockLine1
  output 485, 495
BlockLine2
  output 485
BlockLine3
  output 485
BlockLine4
  output 485
BlockLine5
  output 485
BlockLine6
  output 485
BlockRight
  output 507, 538, 567
BlockSuffix
  output 485, 529, 558
BlockSuffixLeft
  output 507, 538, 567
BlockSuffixRight
  output 508, 538, 567
boundary files 650
buffering 653
Build Global Address 245
Building
  output 253, 261, 285
BuildingSearch
  input option 475
Business Steward 904

C

C API
  Constants 64
  DataRow 95
  DataTable 88
  Error messages 65
  Example application 66
  Message 76
  Server 71
  Supported Libraries 58
C language API 21
C++ API
  Constants 116
  Error messages 117
  Example application 118
  Message 125
  Server 121
  Service 124
  Supported Libraries 110
C++ language API 21
Calculate Distance 694
CAN.Census_CD
  output 425
CAN.Census_CMA
  output 425
CAN.Census_CSD
  output 425
CAN.Census_CT
  output 425
CAN.Census_DA
  output 425
Canadian Postal database 723
CanadianDeliveryInstallationAreaName 805, 807
CanadianDeliveryInstallationQualiferName 807
CanadianDeliveryInstallationQualifierName 805
CanadianDeliveryInstallationType 805, 807
CanadianSERPCode 800, 809
CanDeliveryOfficeFormat 797
CanDualAddressLogic 797
CanFrenchApartmentLabel 794
CanLanguage 769
CanNonCivicFormat 795
CanOutputCityFormat 796
CanRuralRouteFormat 796
CanStandardAddressFormat 795
CASS Report Fields
  dialog box 791
  List Name or ID# 791
  List Processor 791
  Mailer Name and Address 791
  Number of Lists 791
CautiousUpdate 278
CBSACode
  output 486, 529, 558
CBSADivisionCode
  output 486, 530, 558
CBSADivisionName
  output 486, 530, 558
CBSAMetro
  output 487, 530, 558
CBSAName
  output 487, 530, 558
Census.BlockCode
  output 668

Census.MatchLevel
output 668
Census.Tract
output 668
CensusBlockID
output 487, 530, 558
CensusTract
output 488, 530, 559
centerline matching 300
CenterlineBearing
output 489
CenterlineBlockLeft
output 489
CenterlineBlockRight
output 489
CenterlineBlockSuffixLeft
output 489
CenterlineBlockSuffixRight
output 489
CenterlineDataCode
output 490
CenterlineDirection
output 490
CenterlineDistance
output 490
CenterlineHouseNumberHigh
output 490
CenterlineHouseNumberLow
output 491
CenterlineIsAlias
output 491
CenterlineLatitude
output 491
CenterlineLeadingDirectional
output 491
CenterlineLongitude
output 491
CenterlineOffset
input option 472
CenterlineParity
output 491
CenterlineRoadClass
output 492
CenterlineSegmentCode
output 492
CenterlineStreetName
output 492
CenterlineStreetSuffix
output 492
CenterlineTrailingDirectional
output 492
centroid matching 301
CentroidPreference
input option 470
centrus database
adding 706
modifying 706
Centrus Enhanced Geocoding 295
Centrus Enhanced Points 298
Centrus Points 298
Centrus Premium Points 298
Centrus TomTom Points 298
City 734, 738, 742, 748, 752, 755, 758, 761, 762, 765, 768, 804, 805, 807, 834, 862, 869, 871, 872
for GetPostalCodes 764
input 466
output 253, 261, 285, 323, 344, 409, 459, 492, 517, 531, 534, 545, 559, 670
Result Codes 817, 877
CityPreferredName
output 500
CityShortName
output 500
CityStateRecordName
output 500
CityType 758, 764
CityWeight 280
Clear
C API 93
C++ API 137
COM 167
Java 200
Client API 21
client timeouts 30
components 26
introduction 26
overview 26
sample applications 29
Secure Socket Layer 29
steps for using 30
Client API Bean Implementation 39
CloseMatchesOnly 318, 338, 398, 457
input option 478
CMRA 828
output 495
codes
address location 575
geographic centroid location 591
match 592
ZIP+4 centroid location 586
Codes
Street centroid location 585
column definitions 716, 717
COM API 21
Constants 150
DataRow 170
DataTable 164

COM API *(continued)*
example application 152
Map 177
Message 157
Server 153
Service 156
CompanyWeight 280
Component Interface 35, 37
Component Object Model (COM) 21, 150
components 19
Confidence
code descriptions 889
for ValidateAddress 811
for ValidateAddressLoqate 874
output 263, 287, 493, 531, 560, 680
configuration
system 717
Connect
COM 154
Java 185
consolidation 14
Constants
.NET API 216
C API 64
C++ API 116
COM API 150
Java API 182
Constructors
for DataRow 139
for DataTable 133
for Message class 125
for Server class 121
context 256
context properties 27
Context properties
.NET API 223
C API 76
C++ API 125
COM API 157
Java API 192
coordinates
single coordinate pair (point) 717
user 717
CoordinateSystem 317, 337, 396, 457, 543
CouldNotValidate 811, 874
Country 245, 249, 735, 738, 742, 748, 752, 755, 760, 761, 763, 765, 768, 804, 806, 808, 862, 869, 871, 872
for GetPostalCodes 764
output 253, 261, 285, 323, 344, 410, 459, 494, 517, 560, 670
Result Codes 817, 878
Country Identifier 304, 904
CountryElevation
output 531
CountryLevel 812
output 503, 535, 563
County
output 459, 517
County.Code
output 668
County.Name
output 669
County.PostalCodeConfidence
output 686
CreateDataRow
C API 97
CreateDataTable
C API 89
CreateMessage
C API 77
CreateServer
C API 71
CrossComponentMatch 278
CrossStreetLeadingDirectional
output 499, 534, 562
CrossStreetName
output 499, 534, 562
CrossStreetSuffix
output 499, 534, 562
CrossStreetTrailingDirectional
output 499, 534, 562
CSACode
output 488, 530, 559
CSAName
output 488, 530, 559
custom database for Enterprise Geocoding 602
CustomerID 769

D

data
DATA token 717
MIF/MID files 717
numeric 716
Data Hub Module 904
Data Normalization Module 904
data sources
geocode address world 429
Database
RDI 729
DatabasePreference 321, 342, 407
databases 20
comma-delimited files 718
converting MDB files 720
converting SHP and DBF files 720

databases *(continued)*
converting TXT and CSV files 719
DPV 298, 724
Enterprise Geocoding Module 294
GeoConfidence 702
importing user-defined spatial 719
Microsoft Access Database files 718
text files 718
Universal Addressing Module 722
user-defined spatial 715
DatabaseSearchOrder 322, 342, 408, 458, 544
DatabaseVersion
output 503, 535, 563
DataRow
C API 95
COM API 170
Java API 203
Dataset 527, 553
DataTable
C API 88
COM API 164
Java API 198
Datum
input option 472
DBF format 718
DebugOutput 271
decimal
degrees 717
DefaultCountry 753
definitions
column 716, 717
point 717
polyline 717
region 717
DeleteDataRow
C API 97
DeleteDataTable
C API 89
DeleteMessage
C API 78
DeleteServer
C API 72
Delivery Point Barcode 775
Delivery Point Validation (DPV) 303, 728
DeliveryPointCode
output 500
Department
output 253, 261, 285
DepartmentName 804, 806
deployment descriptor 35
deployment descriptors 41
Destructor
for DataRow 139
for DataTable 133
Destructor *(continued)*
for Message class 125
for Server class 121
DirectionalMatchingStrictness 746, 790
Disconnect
COM 155
Java 189
Distance
output 531, 560, 696
DPBC
See "Delivery Point Barcode" 775
DPV 828
database 298, 724
output 496
DPVDetermineNoStat 789
DPVDetermineVacancy 790
DPVFootnotes 828
output 496
DPVNoStat 829
DPVSEED Header Record Layout 610, 887
DPVSuccessfulStatusCondition 791
DPVVacant 829
dropped data 830, 884
dual addresses 302
DualAddressLogic 745, 780

E

Early Warning System
database 299, 725
Early Warning System (EWS) 303, 729
ECRCode
output 287
EJB components
building 42
EJB Container
deploying 43
Sun Reference Implementation 43
testing 43
EJB environment variables 42
ejbCreate() 40
ejbRemove 40
Elevation
output 498, 531, 533, 560, 562
eLOT 729
eLOT database 725
Email1
output 287
Email2
output 287
EnableSERP 795
Enterprise Bean Implementation Class 37
enterprise beans 36

Enterprise Designer 21
Enterprise Geocoding Module 905
databases 294
described 294
Geocode US Address 294
Reverse APN Lookup 294
Enterprise Routing Module 905
Get Travel Boundary 614
Get Travel Directions 614
Enterprise Tax Module 905
AssignGeoTAXInfo 648
CalculateDistance 648
described 648
spatial databases 716
utility 719
Error Messages
.NET API 217
C API 65
C++ API 117
COM API 151
Java API 183
error properties 27
Error properties
C API 76
C++ API 125
COM API 157
ESRI shapefile 716
EWS
See also Early Warning System
USPS website 304, 729
See also Early Warning System
EWS download address
USPS website 304, 729
EWSMatch
output 503, 535, 563
Example application
C API 66
C++ API 118
COM API 152
Java API 184
ExpirationDate
output 503, 535, 563
ExtractFirm 786
ExtractUrb 787

F

FailOnCMRAMatch 791
FallbackToGeographic 314, 337, 392
input option 471
FallbackToPostal 314, 337, 392
FallbackToStreet
input option 471
false-positive records
defined 609, 886
DPV header file layout 610, 887
reporting DPV violations 609, 886
reporting LACS/Link violations 612, 889
Federal Information Processing Standards
See FIPS county numbers 774
Field.n.CommitFlag
output 253
Field.n.Index
output 253
Field.n.Name
output 254
Field.n.Value
output 254
FieldIndex 249
output 254
files
MIF/MID 716, 717
FileSearchOrder 658
FindNearestIntersection 554
FindNearestUnranged 554
FindNearsetAddress 553
FIPS county code 526
FIPS county numbers 774
FIPS state code 526
FirmMatchingStrictness 746, 790
FirmName 735, 738, 743, 748, 752, 755, 769, 804, 806, 808, 862, 869, 871, 872
input 466
output 254, 261, 285, 323, 344, 411, 494, 517, 531, 547, 560, 670
Result Codes 817, 878
FirmNameSearch
input option 474
FirstLetterSearch
input option 474
FlagVulgarWords 270
ForceUpdate options 273
FormatComponent options 281

G

G-NAF
range address matching 310
suffix address matching 310
GBR.ALIASED_LOCALITY 349
GBR.AP_STATUS_FLAG 349
GBR.DEPENDENT_LOCALITY 349
GBR.DEPENDENT_STREET_NAME 349
GBR.DOUBLE_DEPENDENT_LOCALITY 349
GBR.HISTORIC_POSTCODE 349
GBR.OSAPR 349

GDT Geocoding 295
Geocode Address Global 350
geocode address world
country coverage 430
data sources 429
geocode address world 427
geographic geocoding 427
geographic ranking 463
postal geocoding 426
single line input 455
Geocode.MatchCode 873
GeocodeLevel 313, 336, 391
Geocoder.MatchCode 597
output 326, 347, 418, 461, 504, 520
geocoding 14
geocoding concepts 300
Geocoding databases 295
GeoConfidence
databases 702
GeoConfidence Module 905
GeoStanMatchScore
output 504, 535, 564
GeoTAX Auxiliary file 649
GeoTAXKey
output 686
GeoTAXKey.MatchCode
output 686
GeoTAXKey.MatchLevel
output 686
Get
C++ 142, 143
Java 205
Get Candidate Address
inputs
AddressLine1 742
AddressLine2 742
AddressLine3 742
AddressLine4 742
AddressLine5 742
City 742
PostalCode 742
StateProvince 742
Get Candidate Addresses Loqate
inputs
AddressLine1 751
AddressLine2 751
AddressLine3 751
AddressLine4 752
City 752
Country 752
FirmName 752
PostalCode 752
StateProvince 752
Get Travel Boundary
input 615
latitude 615
longitude 615
travel boundary cost 615
travelboundary cost units 615
options 616
Get Travel Directions 634
described 634
input 635
route points 636
output 643
distance field 644
distance Units field 644
format field 644
language 644
status 644
status.code 644
status.description 644
time 644
travel options 629, 642
GetByIndex
C API 99
COM 172
GetByName
C API 100
COM 173
GetCandidateAddresses 722
Inputs
DirectionalMatchingStrictness 746
DualAddressLogic 745
FirmMatchingStrictness 746
MaximumResults 745
OutputCasing 745
OutputShortCityName 745
PerformESM 747
StreetMatchingStrictness 746
output
AddressLine1 748
City 748
Country 748
FirmName 748
HouseNumberHigh 748
HouseNumberLow 748
HouseNumberParity 748
PostalCode 749
PostalCode.AddOn 749
RecordType 749
RecordType.Default 749
StateProvince 749
Status 749
Status.Code 749
Status.Description 749
UnitNumberHigh 750

GetCandidateAddresses *(continued)*
output *(continued)*
UnitNumberLow 750
UnitNumberParity 750
USUrbanName 750
GetCandidateAddressesLoqate
Options
DefaultCountry 753
MaximumResults 754
OutputCasing 752
output
AddressLine1 755
City 755
Country 755
FirmName 755
PostalCode 755
PostalCode.AddOn 755
StateProvince 755
Status 739, 755
Status.Code 755
Status.Description 755
outputs
ProcessedBy 755
GetCityStateProvince 722
inputs 756, 757
MaximumResults 758
OutputVanityCity 758
PostalCode 756
output 758
outputs 764
City 758
CityType 758
PostalCode 758
ProcessedBy 759
StateProvince 759
Status 759
Status.Code 759
Status.Description 759, 764
GetCityStateProvinceLoqate
inputs 760
Country 760
MaximumResults 760
PostalCode 760
output 761
outputs
City 761
Country 761
PostalCode 761
ProcessedBy 761
StateProvince 761
Status 761
Status.Code 762
Status.Description 762
GetColumnCount
C API 92
C++ API 136, 142
COM 166, 172
Java 200, 205
GetColumnCountFromRow
C API 99
GetColumnIndex
C API 92
C++ API 136, 141
COM 166, 172
Java 200, 205
GetColumnIndexFromRow
C API 98
GetColumnNames
C API 91
C++ API 135, 141
COM 165, 171
Java 199, 204
GetColumnNamesFromRow
C API 98
GetContext
C API 78
C++ API 126, 127
COM 158
Java 193
GetContextMap
C API 79
COM 159
GetDataRows
C API 93
GetDataTable
C API 88
C++ API 133
COM 164
Java 198
GetError
C API 87
C++ API 132
COM 163
Java 197
GetKey
COM 179
GetLicenseInfo 212
GetOption
C API 83
C++ API 129
COM 161
Java 195
GetOptions
C API 83
C++ API 130
COM 161
Java 196

GetPostalCodes 722
inputs
City 762
Country 763, 765
OutputVanityCity 763
StateProvince 763
Inputs
OutputCityType 763
outputs
City 764
CityType 764
Country 764
PostalCode 764
ProcessedBy 764
StateProvince 764
Status 764
Status.Code 764
GetPostalCodesLoqate
inputs
City 765
StateProvince 765
outputs 766
PostalCodeLoqate 766
ProcessedBy 766
Status 766
Status.Code 766
Status.Description 767
GetRowCount
C API 95
C++ API 138
COM 169
Java 202
GetService
COM 155
Java 190
GetServiceFromServer
C API 74
GetTravelBoundary
output 623
status 623
status.code 624
status.description 624
GetTravelDirections
options 636
GetValue
COM 179
GetVersionInfo 213
Global Sentry Module 905
GNAF PID Location Search 515
GNAFPointType 316, 395, 516
GNISCode
output 669
GovernmentBuilding
output 500
GTX.ErrorCode
output 681
GTX.ErrorDescription
output 681
GTX.WarnCode
output 682
GTX.WarnDescription
output 683

H

Home Interface 35, 36
HomeCountry 250, 259, 266, 779, 865
HouseNumber 738, 806, 808, 871, 872
output 254, 262, 285, 323, 344, 411, 499, 517, 534, 547, 562
Result Codes 818, 878
HouseNumber2
output 499
HouseNumberHigh 748
output 323, 344, 411, 506, 517, 537, 547, 566
HouseNumberLow 748
output 323, 344, 411, 506, 517, 537, 547, 566
HouseNumberParity 748
output 323, 344, 412, 506, 518, 537, 547, 566
HTTP network communications protocol 29

I

identifier
MIF object 716
PLINE 717
in-context results 257
InnerMatchScore 280
input
AddressLine1 465
AddressLine2 465
AddressLine3 465
AddressLine4 465
AddressLine5 465
AddressLine6 465
AddressPreference 302
City 466
FirmName 466
LastLine 466
PostalCode 466
StateProvince 466
input option
AddressPointInterpolation 473
AddressPreference 474
AlwaysFindCandidates 491
BuildingSearch 475
CenterlineOffset 472

input option *(continued)*
CentroidPreference 470
CloseMatchesOnly 478
Datum 472
FallbackToGeographic 471
FallbackToStreet 471
FirmNameSearch 474
FirstLetterSearch 474
KeepCandidates 478
KeepMultimatch 477
MatchMode 478
OutputCasing 481
OutputFields 483
OutputFormattedOnFail 481
OutputPostalCodeSeparator 482
OutputRecordType 482
OutputVerbose 482
PerformDPV 476
PerformLacsLink 476
PreferZipCodeOverCity 477
RetreiveAPN 476
RetreiveElevation 473
Insurance Premium District (IPD) file 650
insurance premium tax 15
International Postal database 723
InternationalCityStreetSearching 801
Intersection
output 504, 535, 564
IPDn.CasualtyFlag
output 672
IPDn.CasualtyRate
output 672
IPDn.ChangeDate
output 671
IPDn.DistrictID
output 671
IPDn.DistrictName
output 671
IPDn.DistrictType
output 671
IPDn.EffectiveDate
output 671
IPDn.ExpirationDate
output 671
IPDn.FireFlag
output 672
IPDn.FireRate
output 671
IPDn.HealthFlag
output 673
IPDn.HealthRate
output 672
IPDn.LifeFlag
output 673
IPDn.LifeRate
output 673
IPDn.MarineFlag
output 672
IPDn.MarineRate
output 672
IPDn.MinimumFlag
output 673
IPDn.MinimumRate
output 673
IPDn.Notes
output 671
IPDn.OtherFlag
output 673
IPDn.OtherRate
output 673
IPDn.UpdateDate
output 671
IPDn.VehicleFlag
output 672
IPDn.VehicleRate
output 672
IPDn.VersionDate
output 671
IPDnBoundaryBuffer.BufferRelation
output 670
IPDnBoundaryBuffer.DistanceToBorder
output 671
IsAlias
output 504, 535, 564
IsCloseMatch
output 326, 347, 418, 461, 504, 520
Iterator
COM 167
Java 201
Iterator method
C++ API 137

J

J2EE environment 39
Java API
Constants 182
DataRow 203
DataTable 198
Error messages 183
Message 192
Server 185
Service 191
Java beans
building 42
Java class
explained 182

Java language API 21
Java object
    explained 182
javax.jms.MessageListener 40

K

KeepCandidates
    input option 478
KeepMultimatch
    input option 477
KeepMultiMatch 317, 338, 397, 457, 777, 782, 867, 868
KeepStandardizationChanges 279

L

LACSAddress
    output 504, 536, 564
LACSLink database 299, 725
LastLine
    input 466
    output 324, 344, 412, 484, 494, 518, 531, 547, 560
Latitude
    output 285, 326, 346, 418, 461, 494, 498, 520, 531, 533, 560, 562, 674, 873
Latitude.Directional
    output 674
latitude/longitude 717
LatLonFormat 555
LatLong
    output 674
LatLong.MatchCode
    output 674
LatLong.MatchLevel
    output 674
LatLong.StreetMatchCode
    output 675
LatLong.StreetMatchLevel
    output 675
LatLongFormat 527, 695
LeadingDirectional 806, 808, 871, 872
    output 324, 345, 412, 499, 518, 534, 547, 563
    Result Codes 818, 879
line
    polyline 717
local interfaces 35
Locality
    output 518
Locatable Address Conversion System (LACS) 303, 729
location codes
    address 575
    geographic centroid 591
    ZIP+4 centroid 586
Location codes
    Street centroid 585
Location Intelligence Module 905
    Closest Site 706
    described 706
    Point In Polygon 706
    spatial databases 716
LocationCode
    output 494, 531, 560
LocationCode.Description
    output 505, 536, 564
LocationPrecision
    output 327, 348, 419, 462, 521
Longitude
    output 285, 326, 347, 418, 461, 494, 498, 520, 531, 533, 560, 562, 675, 873
Longitude.Directional
    output 675

M

Management Console 20
ManagementAPI
    GetLicenseInfo 212
    GetVersionInfo 213
MAP
    COM API 177
master files 648
match codes 592
MatchCode
    output 494, 531, 560
MatchCode.Description
    output 505, 536, 564
MatchMode
    input option 478
MatchScore
    for ValidateAddress 812
    for ValidateAddressLoqate 875
MaxCandidates 318, 338, 397, 457, 544
MaximumResults 251, 260, 735, 745, 754, 758, 760, 782, 868
MCD.Code
    output 669
MCD.Name
    output 669
MCDCode
    output 484
MCDName
    output 484

Merge
C++ API 139, 143
COM 169, 173
Java 203, 206
MergeDataRow
C API 101
MergeDataTable
C API 95
Message
.NET API 223
C API 76
C++ API 125
COM API 157
Java API 192
Message-Driven Bean 35, 39
class requirements 40
implementation 41
Message-Driven Beans
connection parameters 43
MIF/MID
files 717
modules 19
MSA.Code
output 669
MSA.Name
output 669
MultimatchCount 813
MultiMatchCount
output 326, 347, 418, 461, 520
MultipleMatches 813
MustMatch2DigitPostalCode 405
MustMatchCity 319, 340, 400, 480
MustMatchCount 320, 340, 401
MustMatchFullPostalCode 405
MustMatchHouseNumber 318, 339, 399, 479
MustMatchInput 479
MustMatchPostalCode 320, 341, 404, 480
MustMatchPostalDistrict 341, 405
MustMatchStateProvince 320, 341, 402, 480
MustMatchStreet 319, 339, 399, 480

N

name
of column 717
of feature 716
name parsing 14
name standardization 14
name validation 14
named connection
adding 707, 708
deleting 709
modifying 707, 708
named map
adding 711
deleting 713
modifying 711
named table
adding 709
deleting 710
modifying 709
named tile
adding 713
deleting 715
modifying 713
NAVTEQ Geocoding 295
network protocols
HTTP 29
SOCKET 29
NewRow
C API 94
C++ API 138
COM 168
Java 202
Next
COM 178
Normalized address
description 767, 861
number
of columns 717
whole 717
NumberOfCandidateRanges
output 324, 345, 413, 518, 549
NumberOfIPDsFound
output 673
NumberOfRangeUnits
output 324, 345, 413, 518, 549
NumberPAYsFound
output 677
NumberPTCsFound
output 676
NumberPTDsFound
output 679
NumberSPDsFound
output 684
NumberUserBoundariesFound
output 689
numeric data 716

O

object
listing 717
MIF/MID 716
PLINE (polyline) 717
Offset 555

OffsetFromCorner 315, 394, 542
onMessage() 40
option properties 27
Option properties
.NET API 223
C API 76
C++ API 125
COM API 157
Java API 192
out-of-context results 257
OuterMatchScore
output 287
OuterMatchScoreLines 281
output
Action 252
AdditionalInputData 531, 559
AddressLine 484
AddressLine1 252, 260, 284, 323, 344, 409, 492, 517, 531, 545, 559, 669
AddressLine2 252, 261, 284, 323, 344, 409, 492, 517, 531, 545, 559, 669
AddressLine3 252, 261, 284, 670
AddressLine4 252, 261, 284, 670
AddressLine5 252, 261, 284
AddressLine6 252, 261, 284
AddressLine7 261, 284
AddressLine8 252, 261, 284
AddressLineResolved 503
Alternate 506, 537, 566
AlternativeContext 252
AlternativeIndex 252
Alternatives.InContext.Count 252
Alternatives.OutContext 252
Alternatives.OutContext.Count 253
AlternativesInContext 252
ApartmentLabel 253, 261, 284, 323, 344, 409, 499, 517, 533, 545, 562
ApartmentLabel2 499, 534, 562
ApartmentNumber 253, 261, 284, 323, 344, 409, 499, 517, 534, 545, 562
ApartmentNumber2 499, 534, 562
APN 485, 531, 558
AUS.GNAF_CONFIDENCE 328, 421, 522
AUS.GNAF_EIGHT_DIGIT_LATITUDE 328, 421, 523
AUS.GNAF_EIGHT_DIGIT_LONGITUDE 329, 421, 523
AUS.GNAF_GEOCODE_LEVEL 329, 421, 523
AUS.GNAF_PID 329, 422, 524
AUS.GNAF_RELIABILITY 330, 422, 524
AUS.LEVEL_NUMBER 331, 423, 525
AUS.LEVEL_TYPE 331, 424, 525
AUS.MESH_BLOCK_ID 332, 424, 526
AuxiliaryData 484
output *(continued)*
AuxiliaryData.AuxiliaryFile 667
AuxiliaryDAta.StateFile 667
BlockLeft 507, 538, 567
BlockLine 484
BlockLine1 485, 495
BlockLine2 485
BlockLine3 485
BlockLine4 485
BlockLine5 485
BlockLine6 485
BlockRight 507, 538, 567
BlockSuffix 485, 529, 558
BlockSuffixLeft 507, 538, 567
BlockSuffixRight 508, 538, 567
Buiding 285
Building 253, 261
CAN.Census_CD 425
CAN.Census_CMA 425
CAN.Census_CSD 425
CAN.Census_CT 425
CAN.Census_DA 425
CBSACode 486, 529, 558
CBSADivisionCode 486, 530, 558
CBSADivisionName 486, 530, 558
CBSAMetro 487, 530, 558
CBSAName 487, 530, 558
Census.BlockCode 668
Census.MatchLevel 668
Census.Tract 668
CensusBlockID 487, 530, 558
CensusTract 488, 530, 559
CenterlineBearing 489
CenterlineBlockLeft 489
CenterlineBlockRight 489
CenterlineBlockSuffixLeft 489
CenterlineBlockSuffixRight 489
CenterlineDataCode 490
CenterlineDirection 490
CenterlineDistance 490
CenterlineHouseNumberHigh 490
CenterlineHouseNumberLow 491
CenterlineIsAlias 491
CenterlineLatitude 491
CenterlineLeadingDirectional 491
CenterlineLongitude 491
CenterlineParity 491
CenterlineRoadClass 492
CenterlineSegmentCode 492
CenterlineStreetName 492
CenterlineStreetSuffix 492
CenterlineTrailingDirectional 492
City 253, 261, 285, 323, 344, 409, 459, 492, 517, 531, 534, 545, 559, 670

output *(continued)*
CityPreferredName 500
CityShortName 500
CityStateRecordName 500
CMRA 495
Confidence 263, 287, 493, 531, 560, 680
Country 253, 261, 285, 323, 344, 410, 459, 494, 517, 531, 560, 670
CountryLevel 503, 535, 563
County 459, 517
County.Code 668
County.Name 669
County.PostalCodeConfidence 686
CrossStreetLeadingDirectional 499, 534, 562
CrossStreetName 499, 534, 562
CrossStreetSuffix 499, 534, 562
CrossStreetTrailingDirectional 499, 534, 562
CSACode 488, 530, 559
CSAName 488, 530, 559
DatabaseVersion 503, 535, 563
DeliveryPointCode 500
Department 253, 261, 285
Distance 531, 560, 696
DPV 496
ECRCode 287
Elevation 498, 531, 533, 560, 562
Email1 287
Email2 287
EWSMatch 503, 535, 563
ExpirationDate 503, 535, 563
Field.n.CommitFlag 253
Field.n.Index 253
Field.n.Name 254
Field.n.Value 254
FieldIndex 254
FirmName 254, 261, 285, 323, 344, 411, 494, 517, 531, 547, 560, 670
Geocoder.MatchCode 326, 347, 418, 461, 504, 520
GeoStanMatchScore 504, 535, 564
GeoTAXKey 686
GeoTAXKey.MatchCode 686
GeoTAXKey.MatchLevel 686
GNISCode 669
GovernmentBuilding 500
GTX.ErrorCode 681
GTX.ErrorDescription 681
GTX.WarnCode 682
GTX.WarnDescription 683
HouseNumber 254, 262, 285, 323, 344, 411, 499, 517, 534, 547, 562
HouseNumber2 499
HouseNumberHigh 323, 344, 411, 506, 517, 537, 547, 566

output *(continued)*
HouseNumberLow 323, 344, 411, 506, 517, 537, 547, 566
HouseNumberParity 323, 344, 412, 506, 518, 537, 547, 566
Intersection 504, 535, 564
IPDn.CasualtyFlag 672
IPDn.CasualtyRate 672
IPDn.ChangeDate 671
IPDn.DistrictID 671
IPDn.DistrictName 671
IPDn.DistrictType 671
IPDn.EffectiveDate 671
IPDn.ExpirationDate 671
IPDn.FireFlag 672
IPDn.FireRate 671
IPDn.HealthFlag 673
IPDn.HealthRate 672
IPDn.LifeFlag 673
IPDn.LifeRate 673
IPDn.MarineFlag 672
IPDn.MarineRate 672
IPDn.MinimumFlag 673
IPDn.MinimumRate 673
IPDn.Notes 671
IPDn.OtherFlag 673
IPDn.OtherRate 673
IPDn.UpdateDate 671
IPDn.VehicleFlag 672
IPDn.VehicleRate 672
IPDn.VersionDate 671
IPDnBoundaryBuffer.BufferRelation 670
IPDnBoundaryBuffer.DistanceToBorder 671
IsAlias 504, 535, 564
IsCloseMatch 326, 347, 418, 461, 504, 520
LACSAddress 504, 536, 564
LastLine 324, 344, 412, 484, 494, 518, 531, 547, 560
Latitude 285, 326, 346, 418, 461, 494, 498, 520, 531, 533, 560, 562, 674, 873
Latitude.Directional 674
LatLong 674
LatLong.MatchCode 674
LatLong.MatchLevel 674
LatLong.StreetMatchCode 675
LatLong.StreetMatchLevel 675
LeadingDirectional 324, 345, 412, 499, 518, 534, 547, 563
Locality 518
LocationCode 494, 531, 560
LocationCode.Description 505, 536, 564
LocationPrecision 327, 348, 419, 462, 521
Longitude 285, 326, 347, 418, 461, 494, 498, 520, 531, 533, 560, 562, 675, 873

output *(continued)*
Longitude.Directional 675
MatchCode 494, 531, 560
MatchCode.Description 505, 536, 564
MCD.Code 669
MCD.Name 669
MCDCode 484
MCDName 484
MSA.Code 669
MSA.Name 669
MultiMatchCount 326, 347, 418, 461, 520
NumberOfCandidateRanges 324, 345, 413, 518, 549
NumberOfIPDsFound 673
NumberOfRangeUnits 324, 345, 413, 518, 549
NumberPAYsFound 677
NumberPTCsFound 676
NumberPTDsFound 679
NumberSPDsFound 684
NumberUserBoundariesFound 689
OuterMatchScore 287
PAYn.BoundaryBuffer.DistanceToBorder 677
PAYn.DistrictID 677
PAYn.DistrictName 677
PAYn.DistrictType 677
PAYn.ID 677
PAYn.MunicipalEMSTax 677
PAYn.MunicipalIncomeTax 678
PAYn.SchoolDistrictEMSTax 678
PAYn.SchoolDistrictIncomeTax 678
PercentGeocode 532, 560
Place.ClassCode 687
Place.Code 687
Place.IncorporatedFlag 687
Place.LastAnnexedDate 687
Place.LastUpdatedDate 688
Place.LastVerifiedDate 688
Place.Name 688
Place.PostalCodeConfidence 688
POBox 254, 262, 285
PointCode 539
PostalBarCode 501
PostalCode 254, 262, 285, 324, 345, 413, 459, 494, 518, 532, 549, 560, 670
PostalCode.Addon 324, 345, 414, 518, 549
PostalCode.AddOn 254, 262, 285, 494, 532, 560
PostalCode.Base 254, 262, 286, 494, 532, 560
PostalCodeClass 501
PostalCodeExtensionHigh 507, 538, 566
PostalCodeExtensionLow 507, 538, 566
PostalCodeUnique 501
PostalFacility 501
PreAddress 324, 345, 414, 518, 549
Principality 254, 262, 286

output *(continued)*
PrivateMailbox 324, 345, 414, 499, 518, 534, 549, 563
PrivateMailbox.Designator 499, 534, 563
ProcessedBy 494, 532, 560
PTC.MatchCode 676
PTCn.PayrollCode 676
PTCn.PayrollDescription 676
PTCn.PayrollFlag 676
PTDn.BoundaryBuffer.BufferRelation 679
PTDn.BoundaryBuffer.DistanceToBorder 679
PTDn.DistrictID 679
PTDn.DistrictName 679
PTDn.DistrictType 679
PTDn.JurisdictionID 679
PTDn.UpdateDate 679
RecordType 505, 536, 565
RecordType.Default 505, 536, 565
RoadClass 508, 538, 567
RRHC 499, 534, 561
SearchFieldIndex 254
SearchValue 254
SegmentCode 324, 345, 415, 508, 539, 549, 567
SegmentDirection 508, 539, 567
SegmentHouseNumberHigh 508, 539, 567
SegmentHouseNumberLow 508, 539, 567
SegmentLength 508, 539, 567
SegmentParity 324, 345, 415, 508, 518, 539, 549, 567
SessionId 255
SPDn.BoundaryBuffer.BufferRelation 684
SPDn.BoundaryBuffer.DistanceToBorder 685
SPDn.CompiledDate 685
SPDn.DistrictCode 685
SPDn.DistrictName 685
SPDn.DistrictNumber 685
SPDn.EffectiveDate 685
SPDn.UpdateDate 685
SPDn.VersionDate 685
State.Abbreviation 669
StateCode 669
StateCounty 676
StateProvince 255, 262, 286, 460, 494, 519, 532, 561, 670
Status 256, 263, 287, 326, 347, 419, 461, 494, 520, 532, 561, 683, 696
Status.Code 256, 263, 287, 326, 347, 419, 461, 495, 520, 561, 683, 696
Status.Description 256, 263, 287, 327, 348, 419, 462, 495, 521, 532, 561, 683, 696
StreetDataCode 505, 537, 565
StreetDataType 325, 346, 417, 506, 519, 532, 551, 561, 565

output *(continued)*
StreetName 255, 262, 286, 325, 346, 417, 418, 460, 499, 519, 534, 551, 563
StreetPrefix 325, 346, 417, 519, 551
StreetSide 495, 532, 561
StreetSide.NAVTEQ 509
StreetSuffix 255, 262, 286, 325, 346, 417, 500, 519, 534, 551, 563
SubCity 255, 262, 286
SubStreet 255, 262, 286
TrailingDirectional 325, 346, 417, 500, 519, 535, 551, 563
UnitNumberHigh 325, 346, 417, 507, 519, 538, 551, 566
UnitNumberLow 325, 346, 417, 507, 519, 538, 551, 566
UnitNumberParity 507, 538, 566
URL1 288
URL2 288
USBCCheckDigit 501
USCarrierRouteCode 501
USCarrierRouteSort 502
USCityDelivery 502
USCountyName 255, 263, 286, 488, 530, 559
userBoundary(n).BoundaryDescription 689
UserBoundary(n).BoundaryID 689
UserBoundary(n).BufferRelation 689
UserBoundary(n).DistanceToBorder 689
UserBoundary(n).SupplementalBoundaryID 689
USFIPSCountyCode 533
USFIPSCountyNumber 489, 559
USFIPSStateCode 489, 533, 559
USFIPSStateCountyCode 489, 530, 559
USLACS 502
USLACS.ReturnCode 502
USLotCode 503
USUrbanName 495, 533, 561
WCRCode 288
output data
types of 28
OutputAbbreviatedAlias 789
OutputAddressBlocks 773, 864
OutputCasing 259, 528, 556, 745, 752, 777, 865
input option 481
OutputCityType 763
OutputCountryFormat 251, 259, 779, 866
OutputFieldLevelReturnCodes 771, 782, 863, 868
OutputFields 529, 557, 659
input option 483
OutputFormattedOnFail 771, 863
input option 481
OutputMultinationalCharacters 777
OutputPostalCodeSeparator 251, 259, 777
input option 482
OutputPreferredAlias 788
OutputRecordType 528, 557, 770
input option 482
OutputScript/Alphabet 754, 760, 866
OutputShortCityName 745, 778, 793
OutputStreetNameAlias 772, 794
OutputVanityCity 758, 763
OutputVerbose 528, 556
input option 482

P

PAYn.boundaryBuffer.DistanceToBorder
output 677
PAYn.DistrictID
output 677
PAYn.DistrictName
output 677
PAYn.DistrictType
output 677
PAYn.ID
output 677
PAYn.MunicipalEMSTax
output 677
PAYn.MunicipalIncomeTax
output 678
PAYn.SchoolDistrictEMSTax
output 678
PAYn.SchoolDistrictIncomeTax
output 678
Payroll System Tax Code file 651
payroll system tax codes 663
Payroll tax cross-reference files 651
Payroll Tax District (PAY) file 650
PercentGeocode
output 532, 560
PerformASM 784
PerformCanadianAddressProcessing 792, 801
PerformDPV 784
input option 476
PerformESM 747, 784
PerformEWS 785
PerformLacsLink
input option 476
PerformLACSLink 785
PerformLOT 783
PerformRDI 784
PerformSuiteLink 788
PerformUSAddressProcessing 783
personal property tax 15
Personal Property Tax District (PTD) file 650
Place 685

Place.ClassCode
output 687
Place.Code
output 687
Place.IncorporatedFlag
output 687
Place.LastAnnexedDate
output 687
Place.LastUpdatedDate
output 688
Place.LastVerifiedDate
output 688
Place.Name
output 688
Place.PostalCodeConfidence
output 688
PLINE (polyline) 717
POBox
output 254, 262, 285
Result Codes 819, 879
POBoxNumber 806, 808, 871, 872
point
definition 717
Point In Polygon
regions 717
point-level matching 300
PointCode
output 539
Points databases 298
polyline definition 717
polylines (polygons) 717
Postal Addressing Standards
USPS Publication 28 727
PostalBarCode 776, 809
output 501
PostalCode 735, 739, 742, 749, 752, 755, 756, 758, 760, 761, 764, 768, 804, 806, 808, 834, 862, 869, 871, 872
input 466
output 254, 262, 285, 324, 345, 413, 459, 494, 518, 532, 549, 560, 670
Result Codes 819, 880
PostalCode.Addon
output 324, 345, 414, 518, 549
PostalCode.AddOn 739, 749, 755
output 254, 262, 285, 494, 532, 560
PostalCode.Base
output 254, 262, 286, 494, 532, 560
PostalCode.Source 820
PostalCode.Type 821, 881
PostalCodeAddOn 804, 869
PostalCodeBase 804, 869
PostalCodeCity
Result Codes 820
PostalCodeClass
output 501
PostalCodeExtensionHigh
output 507, 538, 566
PostalCodeExtensionLow
output 507, 538, 566
PostalCodeLoqate 766
PostalCodeUnique
output 501
PostalFacility
output 501
PostcodeWeight 280
PreAddress
output 324, 345, 414, 518, 549
PreferPOIOverStreet 406
PreferPostalCodeOverCity 320, 404
PreferZipCodeOverCity
input option 477
Preview tab 245, 614, 635
Principality
output 254, 262, 286
PrivateMailbox 806, 808
output 324, 345, 414, 499, 518, 534, 549, 563
PrivateMailbox.Designator
output 499, 534, 563
PrivateMailbox.Type 806, 808
Process
C++ API 124
COM 157
Java 191
ProcessedBy 739, 755, 759, 761, 764, 766
for ValidateAddress 814
for ValidateAddressLoqate 875
output 494, 532, 560
processing options
setting defaults 28
ProcessMessage
C API 75
PTC.MatchCode
output 676
PTCn.PayrollCode
output 676
PTCn.PayrollDescription
output 676
PTCn.PayrollFlag
output 676
PTDn.BoundaryBuffer.BufferRelation
output 679
PTDn.BoundaryBuffer.DistanceToBorder
output 679
PTDn.DistrictID
output 679
PTDn.DistrictName
output 679

PTDn.DistrictType
output 679
PTDn.JurisdictionID
output 679
PTDn.UpdateDate
output 679
public Message process(Message message) 39
public Message process(Message request) 35
public Service getService(String serviceName) 34
public void connect() 34
public void ejbActivate() 38
public void ejbCreate() 39
public void ejbPassivate() 38
public void ejbRemove() 38, 39
public void setConnectionProperty(String name
String value) 34
public void setSessionContext(SessionContext ic) 38
PutContext
C API 80
C++ API 127, 128
COM 159
Java 194
PutContextMap
C API 81
COM 160
PutOption
C API 84
C++ 130
COM 162
Java 196
PutOptions
C API 85
C++ 131
COM 162
Java 196

R

RDI
definition of 729
RDI Indicator 827
RecordType 749, 814
output 505, 536, 565
RecordType.Default 749, 814
output 505, 536, 565
region definition 717
regions (polygons) 717
remote Interfaces 35
ReplaceAlias options 276
ReportVulgarWords 270
Representational State Transfer (REST) 234
Reset
COM 177
Residential Delivery File processing
definition of 729
Residential Delivery Indicator (RDI) database 726
Result Codes
AddressRecord 815
ApartmentLabel 816, 876
ApartmentNumber 816, 877
City 817, 877
Country 817, 878
FirmName 817, 878
House Number 818, 878
LeadingDirectional 818, 879
POBoxNumber 819, 879
PostalCode 819, 880
PostalCodeCity 820
RRHC 821
RRHC Type 821, 881
StateProvince 821, 881
Street 822, 881
Street Suffix 824, 882
StreetName 823, 882
StreetName.AbbreviatedAlias.Result 822
StreetName.Alias.Type 823
StreetName.PreferredAlias.Result 823
TrailingDirectional 825, 883
USUrbanName 825
RetreiveAPN
input option 476
RetreiveElevation
input option 473
RetrieveAPN 554
RetrieveElevation 528, 556
return codes
ACR 263, 289
ReturnOnlySimilarFirmNames 343, 408
ReturnPOIWithStreet 405
ReturnUnits 695
ReturnUserData 260
reusable deployable components 35
Reverse APN Lookup 526
Reverse Geocode Address Global 539
Reverse Geocoding database 297
RoadClass
output 508, 538, 567
Routing
Time Units field 644
routing database
adding 645
deleting 645
modifying 645
RRHC 806, 808
output 499, 534, 561
Result Codes 821
Type 821, 881

Run Preview 245, 614, 635

S

sales and use tax 15
Sales tax cross-reference files 651
sample code 717
SAP Module 906
SearchDistance 553, 874
SearchFieldIndex
    output 254
SearchValue 250
    output 254
seed record violations, See false-positive records
SegmentCode
    output 324, 345, 415, 508, 539, 549, 567
SegmentDirection
    output 508, 539, 567
SegmentHouseNumberHigh
    output 508, 539, 567
SegmentHouseNumberLow
    output 508, 539, 567
SegmentLength
    output 508, 539, 567
SegmentParity
    output 324, 345, 415, 508, 518, 539, 549, 567
SERP Report Fields 798
    dialog box 798
    ID# 798
    Mailer Name and Address 798
server 19
Server
    .NET 219
    C 71
    C++ 121
    COM 153
    Java 185
ServerConnect
    C API 73
ServerDisconnect
    C API 73
Service
    .NET 222
    C++ 124
    COM 156
    Java 191
Session Bean Methods 38
Session Bean Parts 35
SessionId 245, 247, 250
    output 255
Set
    C++ 144
    Java 206, 208
SetByIndex
    C 102
    COM 174
SetByName
    C API 101
    COM 174
SetConnectionProperty
    C 72
    COM 156
    Java 190
SetContext
    C++ 129
    Java 195
SetContextMap
    C 82
    COM 160
setMessageDrivenContext 40
SetOptions
    C 86
    C++ 132
    COM 163
    Java 197
ShowExtraAddressLine 251, 260
Siebel Module 906
Simple Object Access Protocol (SOAP) 234
single line input
    geocode address world 455
SOCKET network protocol 29
Southern hemisphere 717
spatial databases 715
SPDn.BoundaryBuffer.BufferRelation
    output 684
SPDn.BoundaryBuffer.DistanceToBorder
    output 685
SPDn.CompiledDate
    output 685
SPDn.DistrictCode
    output 685
SPDn.DistrictName
    output 685
SPDn.DistrictNumber
    output 685
SPDn.EffectiveDate
    output 685
SPDn.UpdateDate
    output 685
SPDn.VersionDate
    output 685
Special Purpose District (SPD) file 650
Split indicia 831, 884
Squeeze 555
StandardAddressFormat 778, 781, 782
StandardAddressPMBLine 791, 794
StandardiseComponent options 268

Standardized address
description 767, 861
state-supplied files 649
State.Abbreviation
output 669
StateCode
output 669
StateCounty
output 676
Stateless Session Bean 35, 39
lifecycle of 36
Stateless Session Beans
connection parameters 43
StateProvince 735, 739, 742, 749, 752, 755, 759, 761, 763, 765, 768, 804, 806, 808, 834, 862, 869, 871, 872
for GetPostalCodes 764
input 466
output 255, 262, 286, 460, 494, 519, 532, 561, 670
Result Codes 821, 881
Status 634
for GetCityStateProvince 759
for GetCityStateProvinceLoqate 761
for GetPostalCodes 764
for GetPostalCodesLoqate 766
for ValidateAddress 814
for ValidateAddressLoqate 875
output 256, 263, 287, 326, 347, 419, 461, 494, 520, 532, 561, 683, 696
Status.Code 634, 739, 749, 755
for GetCityStateProvince 759
for GetCityStateProvinceLoqate 762
for GetPostalCodes 764
for GetPostalCodesLoqate 766
for ValidateAddress 814
for ValidateAddressLoqate 875
output 256, 263, 287, 326, 347, 419, 461, 495, 520, 561, 683, 696
outputoutput
Status.Code 532
Status.Description 634, 739, 749, 755
for GetCityStateProvince 759, 764
for GetCityStateProvinceLoqate 762
for GetPostalCodesLoqate 767
for ValidateAddress 815
for ValidateAddressLoqate 876
output 256, 263, 287, 327, 348, 419, 462, 495, 521, 532, 561, 683, 696
Steps for using the Client API 30
Street
Result Codes 822, 881
Street centroid location codes 585
street matching 300
StreetDataCode
output 505, 537, 565
StreetDataType
output 325, 346, 417, 506, 519, 532, 551, 561, 565
StreetMatchingStrictness 746, 790
StreetName 807, 808, 871, 872
Abbreviated alias 822
AliasType 823
output 255, 262, 286, 325, 346, 417, 418, 460, 499, 519, 534, 551, 563
Preferred alias 823
Result Codes 823, 882
StreetPrefix
output 325, 346, 417, 519, 551
StreetSide
output 495, 532, 561
StreetSide.NAVTEQ
output 509
StreetSuffix 807, 808, 871, 872
output 255, 262, 286, 325, 346, 417, 500, 519, 534, 551, 563
Result Codes 824, 882
StreetWeight 280
SubCity
output 255, 262, 286
SubStreet
output 255, 262, 286
SuiteLink database 726
SuiteLinkFidelity 830
SuiteLinkMatchCode 829
SuiteLinkReturnCode 829
Supported Libraries
C++ API 110
Supportied Libraries
C API 58
system configuration 717

T

tax jurisdiction assignment 15
Taxware files 651
TrailingDirectional 807, 808, 871, 872
output 325, 346, 417, 500, 519, 535, 551, 563
Result Codes 825, 883
Transliteration 283
type
column 717
object 717
types of output data 28

U

U.S. addressing concepts 726
U.S. Postal database 723
unique entity identification 14
UnitNumberHigh 750
    output 325, 346, 417, 507, 519, 538, 551, 566
UnitNumberLow 750
    output 325, 346, 417, 507, 519, 538, 551, 566
UnitNumberParity 750
    output 507, 538, 566
Universal Addressing Module 906
    databases 722
    described 722
    GetCandidateAddresses 722
    GetCityStateProvince 722
    GetPostalCodes 722
    Validate Address 722
Universal Name Module 906
URL1
    output 288
URL2
    output 288
USAltAddr 809
USBCCheckDigit 776, 809
    output 501
USCarrierRouteCode 775, 809
    output 501
UsCarrierRouteSort
    output 502
USCityDelivery
    output 502
USCongressionalDistrict 774, 810
USCountyName 774, 810
    output 255, 263, 286, 488, 530, 559
User Auxiliary files 649
user coordinates 717
user-defined boundary files 650
UserBoundary(n).BoundaryDescription
    output 689
UserBoundary(n).BoundaryID
    output 689
UserBoundary(n).BufferRelation
    output 689
UserBoundary(n).DistanceToBorder
    output 689
UserBoundary(n).SupplementalBoundaryID
    output 689
UseReferenceDiacritics 278
UseRelaxedSecondaryMatching 658
UseStateProvidedFile 657
USFinanceNumber 810
USFIPSCountyCode
    output 533
USFIPSCountyNumber 775, 810
    output 489, 559
USFIPSStateCode
    output 489, 533, 559
USFIPSStateCountyCode
    output 489, 530, 559
USLACS 810, 826
    output 502
USLACS.ReturnCode 827
    output 502
USLastLineNumber 810
USLotCode
    output 503
USLOTCode 826
USLOTHex 826
USLOTSequence 826
USPS
    address standards 727
USPS website
    EWS download address 304, 729
USUrbanName 743, 750, 769, 804, 807, 809
    output 495, 533, 561
    Result Codes 825

V

Validate Address 722
    inputs
        AddressLine1 767
        AddressLine2 768
        AddressLine3 768
        AddressLine4 768
        AddressLine5 768
        CanLanguage 769
        City 768
        Country 742, 768
        CustomerID 769
        FirmName 743, 769
        PostalCode 768
        StateProvince 768
        USUrbanName 743, 769
    Inputs
        CanDeliveryOfficeFormat 797
        CanNonCivicFormat 795
        CanOutputCityFormat 796
        CanRuralRouteFormat 796
        CanStandardAddressFormat 795
        EnableSERP 795
        ExtractFrim 786
        ExtractUrb 787
        HomeCountry 779
        InternationalCityStreetSearching 801
        KeepMultiMatch 777, 782

Validate Address *(continued)*
Inputs *(continued)*
MaximumResults 782
OutputAbbreviatedAlias 789
OutputAddressBlocks 773
OutputCasing 777
OutputCountryFormat 779
OutputFieldLevelReturnCodes 771, 782
OutputFormattedOnFail 771
OutputMultinationalCharacters 777
OutputPostalCodeSeparator 777
OutputPreferredAlias 788
OutputRecordType 770
OutputStreetNameAlias 772, 794
PerformDPV 784
PerformLACSLink 785
PerformLOT 783
PerformSuiteLink 788
outputs
AdditionalInputData 803, 805
AddressFormat 811
AddressLine1 803
AddressLine2 803
AddressLine3 803
AddressLine4 804
AddressLine5 804
ApartmentLabel 805, 807, 870, 872
ApartmentLabel2 805
ApartmentNumber 805, 807, 872
ApartmentNumber2 805
CanadianDeliveryInstallationAreaName 805, 807
CanadianDeliveryInstallationQualiferName 807
CanadianDeliveryInstallationQualifierName 805
CanadianDeliveryInstallationType 805, 807
CanadianSERPCode 800, 809
City 804, 805, 807, 872
CMRA 828
Confidence 811
CouldNotValidate 811
Country 804, 806, 808, 872
CountryLevel 812
DepartmentName 804, 806
DPV 828
DPVFootnotes 828
DPVNoStat 829
DPVVacant 829
FirmName 804, 806, 808, 872
HouseNumber 806, 808, 872
IntHexaviaCode 809
IntINSEECode 809
LeadingDirectional 806, 808, 872
Validate Address *(continued)*
outputs *(continued)*
MatchScore 812
MultimachCount 813
MultipleMatches 813
POBoxNumber 806, 808, 872
PostalBarCode 776, 809
PostalCode 804, 806, 808, 872
PostalCodeAddOn 804
PostalCodeBase 804
PrivateMailbox 806, 808
PrivateMailbox.Type 806, 808
ProcessedBy 814
RDI Indicator 827
RecordType 814
RecordType.Default 814
RRHC 806, 808
StateProvince 804, 806, 808, 872
Status 814
Status.Code 814
Status.Description 815
StreetName 807, 808, 872
StreetSuffix 807, 808, 872
SuiteLinkReturnCode 829
TrailingDirectional 807, 808, 872
USAltAddr 809
USBCCheckDigit 776, 809
USCarrierRouteCode 775, 809
USCongressionalDistrict 774, 810
USCountyName 774, 810
USFinanceNumber 810
USFIPSCountyNumber 775, 810
USLACS 810, 826
USLastLineNumber 810
USLOTCode 826
USLOTHex 826
USLOTSequence 826
USUrbanName 804, 807, 809
Validate Address AUS
inputs
AddressLine1 834
AddressLine2 834
AddressLine3 834
AddressLine4 834
City 834
PostalCode 834
StateProvince 834
Validate Address Loqate
inputs
AddressLine1 862
AddressLine2 862
AddressLine3 862
AddressLine4 862
City 862

Validate Address Loqate *(continued)*
inputs *(continued)*
Country 862
FirmName 862
PostalCode 862
StateProvince 862
Inputs
HomeCountry 865
KeepMultiMatch 867, 868
MaximumResults 868
OutputAddressBlocks 864
OutputCasing 865
OutputCountryFormat 866
OutputFieldLevelReturnCodes 863, 868
OutputFormattedOnFail 863
Output
OutputScript/Alphabet 754, 760, 866
outputs
AdditionalInputData 869
AddressLine1 869
ApartmentNumber 870
ApartmentNumber2 870
City 869, 871
Confidence 874
CouldNotValidate 874
Country 869, 871
FirmName 869, 871
Geocode.MatchCode 873
HouseNumber 871
LeadingDirectional 871
MatchScore 875
POBoxNumber 871
PostalCode 869, 871
PostalCodeAddOn 869
PostalCodeBase 869
ProcessedBy 875
SearchDistance 874
StateProvince 869, 871
Status 875
Status.Code 875
Status.Description 876
StreetName 871
StreetSuffix 871
TrailingDirectional 871
Validate Global Address 264, 840
ValidateAddress
Inputs
DirectionalMatchingStrictness 790
DPVDetermineNoStat 789
DPVDetermineVacancy 790
DPVSuccessfulStatusCondition 791
DualAddressLogic 780
FailOnCMRAMatch 791
FirmMatchingStrictness 790
OutputShortCityName 778, 793
PerformASM 784
PerformESM 784
PerformEWS 785
PerformRDI 784
ReturnVerimove 790
StreetMatchingStrictness 790
ValidateComponent options 271
Vertex files 651
Visual Basic
COM examples 150

W

WADL 237
WCRCode
output 288
Web API
WADL 237
WSDL 235
Web Application Description Language 237
web services 234, 235
SOAP request and response 235
third-party integration tools 234
XML request and response 238
Web Services API 21
Web Services Description Language 235
Western hemisphere 717
WSDL 235

Z

ZIP + 4 Centroid 295